INTRODUCCIÓN AL USO DE LOS STATECHARTS PARA EL DISEÑO DE SISTEMAS EMBEBIDOS

MARIANO BARRÓN RUIZ
Departamento de Ingeniería de Sistemas y Automática. Universidad del País Vasco. España.

Este documento detalla el procedimiento de diseño de sistemas complejos basados en microcontrolador mediante statecharts, utilizado en las clases de la asignatura optativa de Ingeniería Técnica en Electrónica Industrial "Desarrollos con microcontroladores" impartida en la EUITI de Eibar. El documento incluye un ejemplo de diseño con statecharts, utilizado en la asignatura, y se describe la organización, los contenidos, la metodología y las herramientas utilizadas

1. Introducción

Una máquina de estados es un modelo computacional, basado en la teoría de autómatas, que se utiliza para describir sistemas cuyo comportamiento es función de los eventos actuales y de los eventos que ocurrieron en el pasado. En cada instante de tiempo la máquina se encuentra en un estado, y dependiendo de las entradas, actuales y pasadas, que provienen del ambiente, la máquina cambia, o no, de estado pudiendo realizar acciones que a su vez influyen en el ambiente. Las máquinas de estados tradicionales son una excelente herramienta para el tratamiento de problemas sencillos, pero su utilidad disminuye con problemas de moderada complejidad y resultan prácticamente inútiles en la descripción de sistemas complejos.

Durante la década de 1980 David Harel [1,2] propuso una amplia extensión al formalismo convencional de las máquinas y diagramas de estados a la que denominó *statecharts*. El término, según palabras de su autor, fue elegido por ser una combinación no utilizada del las palabras "flow" o "state" con "diagram" o "chart". El objetivo principal del nuevo formalismo visual era, y lo sigue siendo, modelizar o permitir la descripción de *sistemas reactivos* [3] cuyo comportamiento puede llegar a ser tan complejo como para que la plasmación de sus especificaciones resulte muy difícil y propensa a errores. El término *reactivo* se aplica a objetos que responden dinámicamente a los eventos de interés que reciben, y cuyo comportamiento lo determina el orden de llegada de esos eventos. Constituyen ejemplos de sistemas reactivos: los cajeros automáticos, los sistemas de reservas de vuelos, los sistemas embebidos en aviones y automóviles, los sistemas de telecomunicaciones, los sistemas de control, etc.

Los statecharts extienden los diagramas de transición de estados convencionales con tres elementos principales: jerarquía, concurrencia y comunicación. El uso de jerarquías permite tratar los sistemas con diferentes niveles de detalle; la concurrencia, también llamada ortogonalidad y paralelismo, posibilita la existencia de tareas independientes entre si o con escasa relación entre ellas, y la comunicación hace viable que varias tareas reaccionen ante un mismo evento o envíen mensajes hacia otras tareas.

El uso de los statechart de Harel ha crecido considerablemente desde que una variante de los mismos se ha convertido en uno de los diagramas utilizados por UML (Unified Modeling Language) [4] para describir el comportamiento de sistemas o de modelos abstractos. UML considera los diagramas gráficos como vistas o representaciones parciales del modelo de un objeto; los diagramas de UML representan tres vistas distintas del modelo: la vista de sus necesidades funcionales, la vista de su estructura y la vista de su comportamiento. La versión 2.0 de UML contempla el uso de hasta 13 tipos de diagramas que enfatizan la estructura, el comportamiento y la interacción entre las partes de un sistema.

2. Herramientas UML Statecharts dedicadas al diseño de sistemas embebidos

Aunque es posible modelizar sistemas reactivos sin la ayuda de herramientas CASE, tal como propone el autor Miro Samek [5], lo cierto es que estas herramientas facilitan el trabajo y aportan otros aspectos importantes como: sus cómodos interfaces gráficos; la posibilidad de disponer rápidamente de un modelo claro y ejecutable que permita la simulación temprana del comportamiento del sistema; la verificación funcional del modelo; la generación automática de código C, C++, Java, Ada; la generación automática de documentación; el seguimiento del grado de cumplimiento de las especificaciones; etc. Son numerosas las herramientas comerciales [6,7,8,9] disponibles tales como Rational Rose o Telelogic Rhapsody, así como las herramientas libres o incluso herramientas open source como IntelliWizard [10].

Una herramienta comercial especialmente adaptada al diseño de estos sistemas es visualSTATE, creada por la compañía sueca IAR Systems [11] dedicada a la creación de software para desarrollo de sistemas embebidos que soporta diferentes familias de microcontroladores. Aunque visualSTATE no es una herramienta UML, ya que solo contempla el uso de los diagramas statecharts, es la herramienta seleccionada por nosotros para introducir los statecharts en el diseño de sistemas embebidos. Las razones que justifican su elección son: su sencillez de uso, su eficiencia y la disponibilidad de una versión demo con toda la funcionalidad de la versión comercial, pero limitada a 20 estados, que resultan suficientes para su uso en la Escuela. La sencillez de uso se debe a que no exige aprender los 13 diagramas soportados por UML 2.0 sino que basta con aprender un solo diagrama, el de los statecharts. La eficiencia se debe a la capacidad de visualSTATE para generar código muy compacto, de tamaño considerablemente menor que el generado por otras herramientas UML como Telelogic Rhapsody. El código generado por visualSTATE es tan compacto que sistemas de baja y mediana complejidad caben perfectamente en pequeños microcontroladores de 8 bits provistos de tan solo 2Kbytes de memoria de código. En el lado negativo debemos anotar, entre otros aspectos, que la herramienta no soporte la fase de análisis, que no proporcione diagramas estructurales y que tampoco incluya un seguimiento de las especificaciones de los sistemas.

3. Características de la herramienta visualSTATE

visualSTATE es un entorno gráfico para diseño, verificación e implementación de sistemas embebidos basados en máquinas de estados jerárquicas o statecharts. Entre sus características destacan:

- Un entorno de desarrollo integrado que incluye un editor gráfico, herramientas de verificación y simulación, un generador automático de código C y/o C++, y un generador automático de documentación.
- Diseño gráfico de máquinas de estados jerárquicas basado en el subconjunto UML-Statechart.
- Verificación formal o matemática del modelo para hallar propiedades no deseadas del diseño tales como: estados sin salida, estados inalcanzables, etc.
- Herramienta de simulación o validación que permite, desde las primeras etapas del diseño, verificar que la aplicación se comporta de la forma deseada.
- Generador automático de código C/C++. El código generado es muy compacto y conforme al 100% con el modelo validado.
- Generador automático de documentación en formato RTF o HTML.

4. Ventajas del diseño dirigido por modelos

Los statecharts permiten construir modelos gráficos que describen con precisión el comportamiento de un sistema. Los modelos creados no tienen ninguna relación con el lenguaje de programación que vaya a utilizarse en el desarrollo de la aplicación, sin embargo, si tienen una relación muy estrecha con el funcionamiento deseado de la aplicación. Esta relación facilita la comunicación y el intercambio de ideas entre el cliente y el equipo de desarrollo del sistema, independientemente del tipo de formación que posean los miembros del equipo. El modelo permite simular y visualizar la aplicación desde las primeras

etapas del diseño sin necesidad de construir un prototipo hardware; esta característica facilita la eliminación de errores desde el principio. Los programadores deben de cambiar la forma tradicional en la que abordan la tarea de desarrollo de software trasladando su forma de pensar al dominio de la aplicación y liberándose de las limitaciones impuestas por el lenguaje de programación utilizado. Si la herramienta de modelado dispone de generadores automáticos de código y de documentación los beneficios son aún mayores, ya que los cambios que se realizan y simulan en el modelo se trasladan automáticamente al código generado y a la documentación generada, por lo que la propia herramienta se encarga de mantener el sincronismo entre el modelo, el código y la documentación. Disponer de documentación actualizada es de un aspecto de enorme importancia ya que facilita el mantenimiento de las aplicaciones.

5. Ejemplo de diseño con statecharts

En este apartado se describe uno de los primeros ejemplos utilizados en clase para introducir el diseño de sistemas embebidos mediante statecharts. El ejemplo ilustra algunos elementos importantes de los statecharts, aunque no todos, y muestra la nueva metodología de diseño que partiendo de las especificaciones termina generando el firmware para el microcontrolador utilizado.

5.1. Especificaciones del sistema

Se trata de diseñar un controlador de un sencillo horno microondas provisto de grill, una luz interior, dos pulsadores Start/Stop y Modo, y un conmutador para monitorizar el estado de la puerta abierta/cerrada. Las especificaciones son las siguientes:

1. La luz interior debe encenderse cuando se abra la puerta y cuando funcione el horno o el grill.
2. Existe un botón Start/Stop para conmutar entre los modos de reposo y de funcionamiento.
3. Al abrir la puerta el horno debe dejar de funcionar. Cuando se cierre la puerta el horno deberá regresar al modo de trabajo o reposo en que se encontrara antes de abrir la puerta.
4. El horno posee un segundo botón para seleccionar el modo de trabajo. Por cada pulsación de este botón, el modo de trabajo cambia cíclicamente entre Horno, Grill y Horno + Grill.
5. Para indicar el modo de trabajo el horno dispone de tres leds: H, G y H+G

5.2. Primer paso. Identificar los eventos y las acciones

Los *eventos* representan la influencia del ambiente sobre el sistema y serán las entradas a la máquina de estados. En nuestro ejemplo los eventos serán los cuatro siguientes:

Nombre del evento	**Producido cuando el usuario:**
eModo	Pulsa el botón de Modo
eStartStop	Pulsa el botón de Start/Stop
ePuertaAbrir	Abre la puerta del horno
ePuertaCerrar	Cierra la puerta del horno

Las *acciones* representan la influencia del sistema sobre el ambiente y serán las salidas de la máquina de estados. Las acciones se realizan mediante llamadas a funciones escritas en lenguaje C.

Acción	**Trabajo realizado**
aLuzOn(void)	Enciende la luz interior del horno
aLuzOff(void)	Apaga la luz interior del horno
aLED(unsigned char uc)	Enciende el led H si uc=1, el led G si uc=2, o el led H+G si uc=3
aEnciende(unsigned char uc)	Enciende el Horno si uc=1, el Grill si uc=2, el Horno y el Grill si uc=3. Apaga el Horno y el Grill si uc = 0

5.3. Segundo paso. Identificar los estados

Un *estado* es una condición o situación durante la vida de un objeto en la que se satisface alguna condición, se realiza alguna actividad, o se espera algún evento. Los estados pueden identificarse a partir de las especificaciones y del conocimiento del problema. En nuestro ejemplo los estados, que se representan por rectángulos con los bordes redondeados, pueden ser los de la figura 1: la puerta puede estar Abierta o Cerrada, el horno puede estar Cocinando o Esperando y el modo de trabajo puede ser Horno, Grill y Horno+Grill.

Figura 1. Identificación de los estados.

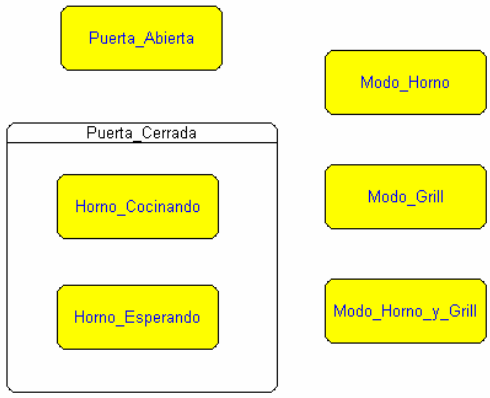

Figura 2. Agrupamiento por jerarquías.

5.4. Tercer paso. Agrupar por jerarquías

En este paso se trata de determinar los estados que tienen un comportamiento dinámico propio y los estados que sólo pueden estar activos en ciertas situaciones. El agrupamiento de estados de la figura 2 indica que las situaciones Horno_Cocinando y Horno_Esperando sólo tienen sentido si la puerta del horno está cerrada. El estado Puerta_Cerrada es un estado compuesto (a veces llamado superestado) que tiene dos estados hijo: Horno_Cocinando y Horno_Esperando. Cuando el horno se encuentre con la puerta cerrada, la máquina podrá estar en uno de los dos estados hijo pero no en ambos a la vez.

5.5. Cuarto paso. Agrupar por concurrencia

Organizar el modelo en varias máquinas de estados paralelas después de examinar los estados que pueden estar activos al mismo tiempo.

En nuestro ejemplo se debe poder cambiar el modo de funcionamiento con el horno esperando o cocinando y tanto si la puerta está abierta como si está cerrada. En consecuencia se usan dos máquinas concurrentes o paralelas. La concurrencia se representa por regiones separadas por una línea vertical discontinua como puede apreciarse en la figura 3.

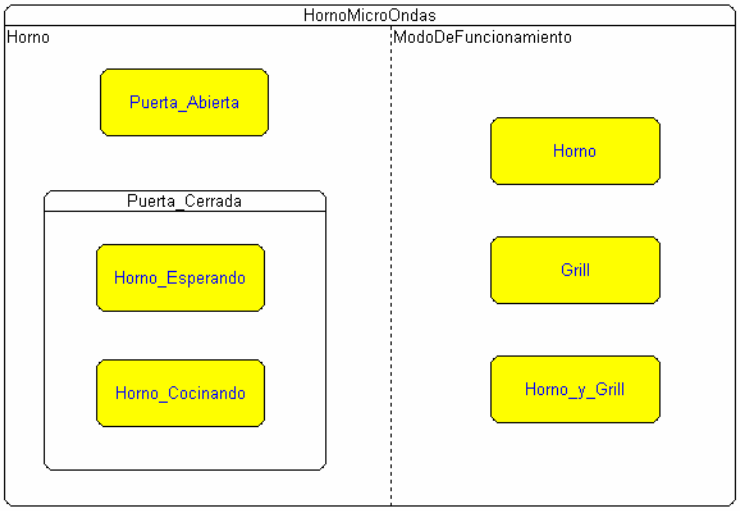

Figura 3. Agrupamiento por concurrencia.

5.6. Quinto paso. Añadir las transiciones

En este paso hay que identificar las acciones a realizar y los cambios de estados que se deben producir tras un evento. Las transiciones se representan por flechas dirigidas desde el estado origen hacia el estado destino. La figura 4 muestra la máquina de estados después de haber añadido las transiciones y las sincronizaciones correspondientes al sexto y último paso del diseño que se trata en el siguiente sub-apartado. El evento eModo dispara la transición desde el estado H hacia el G, desde el G hacia el HyG, y desde éste último hacia el estado H. El evento eStartStop dispara la transición desde el estado Esperando hacia el estado Cocinando y desde el estado Cocinando hacia el estado Esperando.

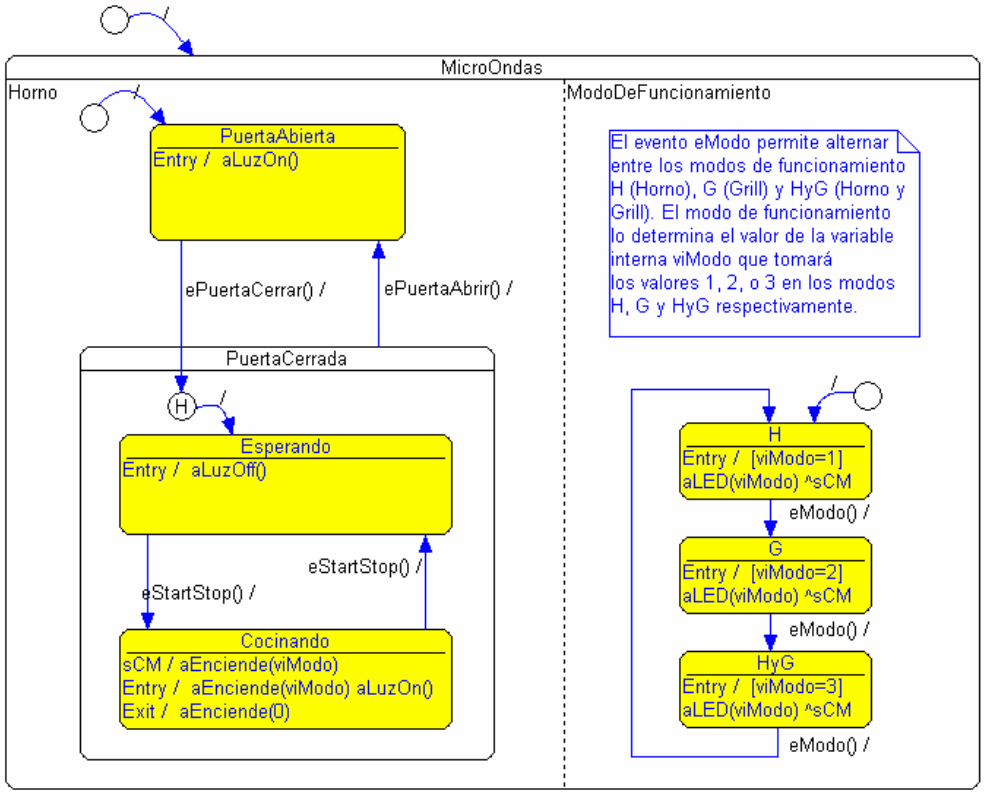

Figura 4. Statechart completo obtenido después de añadir las transiciones y las sincronizaciones.

El evento ePuertaAbrir dispara la transición desde el estado PuertaCerrada.Esperando o desde el estado PuertaCerrada.Cocinado hacia el estado PuertaAbierta. El evento ePuertaCerrar dispara la transición desde el estado PuertaAbierta hacia el *pseudo-estado historia superficial* representado por un pequeño círculo con una H en su interior. Un pseudo-estado representa un estado transitorio en el que una máquina de estados no puede permanecer; cuando la máquina alcanza un pseudo-estado se ejecuta de forma automática la transición de salida de ese pseudo-estado. La primera vez que se sale desde el *pseudo-estado historia superficial* la transición se dirige hacia el estado Esperando, sin embargo, las sucesivas transiciones de salida desde este pseudo-estado se dirigen hacia el estado hijo en el que se encontraba el sistema antes de abandonar el estado padre PuertaCerrada. Este comportamiento se corresponde con la especificación número 3 del sub-apartado 5.1 “Cuando se cierre la puerta el horno deberá regresar al modo de trabajo o reposo en que se encontrara antes de abrir la puerta.”.

En la figura 4 aparecen otros tres pseudo-estados representados por un pequeño círculo sin nada en su interior. Se trata del *pseudo-estado default* origen de la transición inicial; así, al iniciar la máquina de

estados se entra al estado compuesto MicroOndas que está situado en el nivel jerárquico más alto. Este estado compuesto contiene dos máquinas de estados concurrentes, Horno y ModoDeFuncionamiento cuya ejecución corre de forma paralela. Cada una de estas dos máquinas debe arrancar en un estado conocido que viene definido por cada transición inicial; de esta forma la máquina Horno se inicia en el estado PuertaAbierta y la máquina ModoDeFuncionamiento se inicia en el estado H.

Las reacciones Entry presentes en todos los estados con fondo amarillo y la reacción Exit, presente sólo en el estado Cocinando, son *reacciones internas* que se ejecutan automáticamente al entrar (Entry) o al salir (Exit) al/del estado correspondiente. Al entrar en el estado PuertaAbierta se ejecuta la función de acción aLuzOn() asociada a la reacción Entry, que enciende la luz interior del horno. Al salir del estado Cocinando se ejecuta la función de acción aEnciende(0) asociada a la reacción Exit. Esta función al ser llamada con un parámetro de valor 0 apaga el Horno y el Grill tal como se reflejó en la tabla de las funciones de acción del sub-apartado 5.2.

5.7. Sexto paso. Añadir las sincronizaciones

Las sincronizaciones son los mensajes internos que una máquina de estados puede enviar a otra máquina. En visualSTATE, los mensajes que una máquina envía hacia otra máquina se llaman señales. Las señales, al igual que los eventos, pueden disparar nuevas transiciones. En la figura 4 se muestra la señal de cambio de modo sCM, que la máquina de estados ModoDeFuncionamiento envía hacia la máquina Horno. Cada vez que el evento eModo dispara una transición: se entra en el estado H, o G o HyG; la reacción Entry de cada estado asigna el valor 1, 2 o 3 a la variable interna viModo; se enciende el LED cuyo número coincide con el valor de la variable viModo y se genera la señal sCM. La máquina Horno solo es receptiva a la señal cambio de modo si se encuentra en el estado Cocinando; en este caso la señal sCM dispara una reacción interna y llama a la función aEnciende(viModo) que conectará el Horno, el Grill, o ambos, dependiendo del valor de la variable viModo.

El statechart de la figura 4 contiene un modelo verificable del horno microondas propuesto. Este modelo es todo lo que se necesita para simular el sistema y comprobar si su funcionamiento es conforme a las especificaciones. El ejemplo descrito sirve para valorar la enorme capacidad descriptiva de los statechart y el nivel de abstracción al que se trabaja en los diseños dirigidos por modelos. Para generar el firmware del microcontrolador además del modelo anterior se necesitan las funciones de acción, el manejador de la cola de eventos, la función main() y las funciones necesarias de la biblioteca de visualSTATE. Todo este código adicional sólo se escribe una vez y es casi idéntico (salvo las funciones de acción) para cualquier sistema.

6. Código requerido por una aplicación creada con visualSTATE

La modelización de un sistema, por medio de statecharts, permite validar el diseño de forma interactiva desde que comienza el proceso, hasta que se consideran cubiertas satisfactoriamente todas las especificaciones del sistema. Durante este proceso de diseño y validación no es necesario generar código C, ni disponer de ningún hardware, únicamente se necesita trabajar con las herramientas *Designer* y *Validator* de visualSTATE. Una vez terminada la fase de diseño y validación interactiva, debe seguirse un proceso de verificación formal con objeto de comprobar, de forma automática, la consistencia lógica del proyecto; la verificación formal del modelo la realiza el programa *Verificator* de visualSTATE. Tras esta fase llega el momento de generar código C para programar el microcontrolador. La parte más importante del código la genera automáticamente la herramienta *Coder* de visualSTATE. El comportamiento del código generado es idéntico al del modelo que se ha validado, sin embargo, no todo el código que necesita la aplicación se genera de forma automática, el diseñador también tiene que escribir manualmente una parte (menor) del código.

Para crear una aplicación con visualSTATE se necesita código de tres tipos:

1. Código generado automáticamente por visualSTATE
2. La API (Application Programming Interface) de visualSTATE
3. Código generado por el usuario

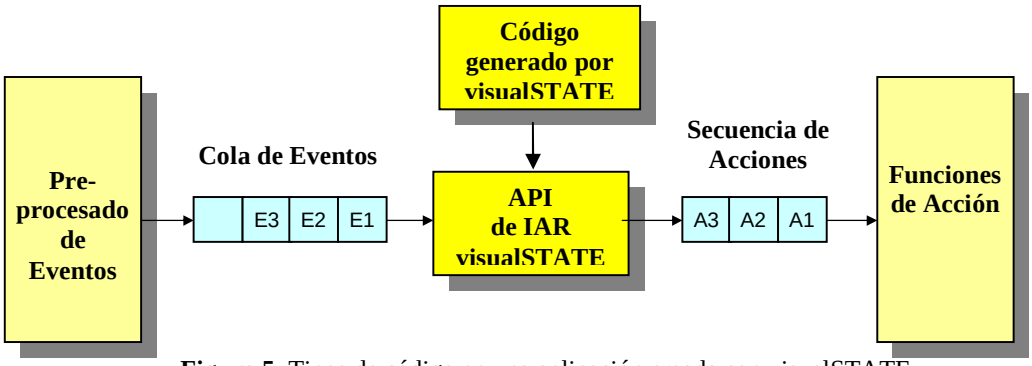

Figura 5. Tipos de código en una aplicación creada con visualSTATE.

Dado que los dos primeros tipos de código los proporciona visualSTATE, el diseñador sólo tiene que escribir manualmente el siguiente código:

- Código para inicializar el hardware
- Código para procesar las entradas (generar los eventos y manejar la cola de eventos)
- Código para procesar los dispositivos de salida (funciones de acción)
- La función main

visualSTATE proporciona ejemplos que contienen código fuente para el manejo de la cola de eventos y el código típico de la función main; el diseñador puede adaptar este código de ejemplo y limitarse a escribir las funciones que convierten los cambios de las entradas en eventos y las funciones que actúan sobre las salidas del sistema, también llamadas funciones de acción. En general las funciones escritas por el usuario sólo se escriben una vez para cada aplicación. La función main normalmente comienza con la inicialización de los periféricos del microcontrolador, del sistema visualSTATE y de la cola de eventos; posteriormente la función entra en un bucle sin fin durante el cual examina la cola de eventos y en el caso de que no esté vacía, extrae el primer evento almacenado, ejecuta las funciones de acción asociadas al evento y dirige la máquina de estados hacia el siguiente estado especificado en el statechart. El procesado de los eventos se realiza respetando el orden en que se han ido produciendo.

Una vez conseguido el funcionamiento deseado es conveniente utilizar la herramienta *Documenter* de visualSTATE para generar automáticamente un informe actualizado del proyecto, que puede incluir información sobre: diseño, validación, simulación, verificación, código generado e implementación. Dentro del *Documenter*, el diseñador puede especificar el tipo de información que desea incluir en el informe así como el formato de salida del mismo (RTF o HTML).

Si posteriormente es necesario añadir nuevas prestaciones al sistema, los cambios se realizan nuevamente trabajando con los statecharts dentro de visualSTATE y siguiendo el mismo proceso que se ha descrito en los apartados anteriores. Finalmente se termina generando código de forma automática e integrándolo con el código escrito manualmente por el diseñador. En ningún momento es necesario retocar a mano el código generado por visualSTATE, tampoco es una práctica aconsejable y además, al evitarla, se garantiza el sincronismo entre el modelo, el código y la documentación.

7. Desarrollo de la asignatura

La introducción a los statecharts para el diseño de sistemas embebidos, es parte del contenido de la asignatura optativa *Desarrollos con microcontroladores*, ofertada a los estudiantes de Ingeniería Técnica en Electrónica Industrial en la EUITI de Eibar (Guipúzcoa); se trata de una asignatura de 6 créditos que se imparte a lo largo del segundo cuatrimestre. Los alumnos que eligen la asignatura generalmente disponen de conocimientos básicos de microcontroladores y de programación en lenguaje C, además manejan con cierta soltura el programa de captura de esquemas y simulación electrónica PROTEUS [12, 13].

La asignatura se desarrolla durante 15 semanas, a razón de dos sesiones semanales de 2 horas, en el Laboratorio de Microelectrónica dotado de ordenadores en todos los puestos de trabajo y de un proyector de imágenes para el PC del profesor. Como elemento de apoyo a la docencia presencial se utiliza la plataforma Moodle [14], donde se depositan los recursos didácticos, se recogen las tareas asignadas a los alumnos, se muestran las calificaciones, y se dispone de un foro para realizar labores de tutoría virtual. Las clases comienzan con la exposición de un tema, utilizando una presentación en PowerPoint, a la que le sigue uno o dos ejercicios prácticos realizados en el PC. Cada semana, o cada dos semanas, se enuncia una tarea que los alumnos deben realizar en sus casas y subirla a la plataforma Moodle, antes de la primera clase de la siguiente semana. De este modo, la primera clase de cada semana se inicia con el examen crítico y discusión de la solución adoptada por algún o algunos alumnos a la tarea encargada. El proceso seguido en las clases es: exposición de un tema – práctica – trabajo individual – comentario crítico. Todas las tareas se califican, y se tienen en cuenta para la nota final de la asignatura.

El objetivo de la asignatura es desarrollar el hardware y el software de sistemas prácticos basados en microcontrolador, por lo tanto es lógico que el trabajo a realizar durante los ejercicios prácticos y en las tareas, consista en: capturar el hardware de los sistemas en un esquema, crear el firmware utilizando un entorno de desarrollo en lenguaje C y depurar el software hasta conseguir que el sistema cumpla las especificaciones. El funcionamiento del sistema se verifica con la ayuda de un simulador. El microcontrolador seleccionado para las prácticas es el AVR ATmega16 de Atmel [15], elegido por sus prestaciones, por la cantidad de periféricos que incorpora y porque el software de simulación utilizado es capaz de simular el microcontrolador y todos sus periféricos, lo que agiliza la realización de trabajos en clase y en casa.

7.1 Herramientas software utilizadas

Todas las herramientas utilizadas en la asignatura pueden obtenerse gratuitamente, salvo el software PROTEUS, que también se utiliza en otras asignaturas y para el cual se dispone de licencias. Algunas de las herramientas son versiones demo de software profesional que resultan suficientes para su uso en un entorno escolar. El listado completo de las herramientas es el siguiente:

- Proteus 7 Professional [12], para captura de esquemas, simulación y ruteado
- CodeVisionAVR Evaluation [16], Entorno integrado de desarrollo, compilador C, Generador automático de código de inicialización de periféricos y programador ISP
- AVR Studio 4 [15], Entorno integrado de desarrollo profesional, para escritura y depuración de aplicaciones con microcontroladores AVR
- IAR visualSTATE 20-state evaluation edition [11]

7.2 Documentos básicos

Los documentos más utilizados a lo largo del curso son:

- La hoja de datos del microcontrolador ATmega16
- El set de instrucciones de la familia de microcontroladores de 8-bits AVR
- El manual de usuario del compilador CodeVisionAVR

7.3 Contenido del curso

El curso se estructura en 12 temas:

1. Lenguaje C adaptado a los microcontroladores AVR
2. Arquitectura y características generales de la familia de µC de 8-bits AVR
3. Los puertos de I/O
4. Las interrupciones del ATmega16
5. Los timers&counters del ATmega16
6. Programación de visualizadores LCD alfanuméricos
7. Exploración de conmutadores y teclados. Supresión de rebotes
8. Comunicación serie RS-232, RS-485, SPI e I^2C
9. El comparador analógico y el ADC del ATmega16
10. Fuentes de reset en el ATmega16. Watchdog
11. Consumo de energía. Modos SLEEP
12. Diseño con stateCharts

A lo largo del cuatrimestre se completan, entre trabajos en clase y tareas para casa, más de 20 ejercicios completos de diseño (hard+soft). Solo el estudio de todos los periféricos del AVR y la realización de los ejercicios que utilizan esos periféricos, exigiría más de un cuatrimestre, en caso de no disponer de herramientas de diseño de alta productividad. En este sentido cabe destacar la aportación realizada por la herramienta CodeWizardAVR de CodeVisionAVR; se trata de un asistente para la configuración de periféricos, capaz de crear código C para la inicialización de todos los periféricos de la familia AVR de microcontroladores, que ahorra mucho tiempo de diseño y de depuración. También en el mismo sentido, debemos destacar la ayuda prestada por el software PROTEUS en el apartado de simulación; este software es capaz de realizar simulación digital, analógica y mixta; además, puede simular un microcontrolador, todos sus periféricos internos y todos los dispositivos externos, partiendo del código ejecutable creado por un compilador o un ensamblador; también puede simular PLDs simples, como la PAL22V10 y otras, siempre que se le proporciona un fichero JEDEC para cada PLD. La simulación puede generar gráficos y permite la interacción del usuario con el hardware simulado en tiempo real; cuando el usuario hace clic con el ratón en un teclado simulado, las acciones que se observan son casi las mismas que las del hardware real. Si se trabajase con hardware real, en lugar de hacerlo con un simulador, no sería posible realizar ni una cuarta parte de los ejercicios del curso. Pese a la ayuda aportada por todas estas herramientas, la asimilación de los contenidos del curso requiere más tiempo que el disponible; esta limitación temporal impide tratar el último tema con la extensión deseable. Para poder mostrar las capacidades más interesantes de los statecharts se tiene que recurrir a realizaciones complejas elaboradas por el profesor, en detrimento de las actividades que deberían realizar los alumnos.

8. Conclusiones

La complejidad de software actual y la demanda de ciclos de desarrollo cada vez más cortos, exige la utilización de herramientas de diseño de elevada productividad tales como: lenguajes de alto nivel, asistentes o bibliotecas para la inicialización de periféricos, statecharts, sistemas de generación automática de código, middleware, etc.

Los statecharts están situados en un nivel de abstracción superior al de los lenguajes de programación, lo que les permite conducir el diseño al dominio de la aplicación. Son elementos gráficos provistos de una enorme capacidad descriptiva, son más fáciles de interpretar que los listados de código, resultan muy adecuados para describir comportamientos complejos y simplifican el intercambio de ideas con personas ajenas o no al proyecto. El proceso de desarrollo de sistemas reactivos con statecharts, expuesto en los apartados anteriores, proporciona una eficaz metodología de diseño para programadores

cualquiera que sea su nivel de experiencia. Los statecharts pueden integrarse con RTOS, aunque en muchas aplicaciones pueden reemplazarlos con ventaja, ya que exigen menos código, soportan el paralelismo, permiten la simulación gráfica sin necesidad de compilar ni de disponer de un sistema físico, los errores se detectan con rapidez durante la fase de diseño-validación y vienen acompañados por herramientas de verificación formal, generación automática de código y generación automática de documentación que facilitan notablemente la tarea de los programadores.

Referencias

[1] D. Harel. *Statecharts: a visual formalism for complex systems. Science of Computer Programming.* Vol 8, Nº 3, 231-274 (1987).
[2] David Harel Home page. http://www.wisdom.weizmann.ac.il/~dharel/
[3] D. Harel, A Pnueli. *On the Development of Reactive Systems. Logics and Models of Concurrent Systems.* Vol F-13, 477-498, (1985).
[4] UML Resource Page. http://ww.uml.org
[5] M. Samek. *Practical Statecharts in C/C++*. Elsevier CMPBooks, (2002)
[6] Cetus Links - UML Tools. http://www.cetus-links.org/oo_uml.html#oo_uml_utilities_tools
[7] Mario Jeckle - UML Tools. http://www.jeckle.de/umltools.htm
[8] List of UML tools. http://en.wikipedia.org/wiki/List_of_UML_tools
[9] UML tools. http://www.oose.de/umltools.htm
[10] UML StateWizard. http://www.intelliwizard.com/
[11] IAR Systems Home page. http://www.iar.com
[12] Labcenter Electronics Home page. http://www.labcenter.co.uk/index.cfm
[13] M. Barrón, Uso didáctico del software de ayuda al diseño electrónico "PROTEUS". *Actas del VI Congreso TAEE 2004*, Valencia 2004
[14] Moodle Home page. http://moodle.org
[15] Atmel Corporation Home page. http://www.atmel.com
[16] CodeVisionAVR Home page. http://www.hpinfotech.ro/