Biblioteca de simulación por software multiplataforma para el microcontrolador M68HC11

Olea A. César y Romero A. Eduardo

***Resumen* — En el estudio de sistemas electrónicos es inevitable el tema de programación en lenguaje de bajo nivel para el desarrollo de sistemas basados en microprocesador y microcontrolador. Está claro que es crucial el manejo y empleo de herramientas que faciliten el proceso de enseñanza, tales como los simuladores. Pero con la desventaja de que casi siempre hay que desembolsar un costo por su uso o por la limitante de poder usarse en un sistema operativo específico. De tal forma, sería preferible crear una biblioteca de simulación multiplataforma que tome en cuenta los beneficios de un sistema unificado de clases que permitan el desarrollo de programas capaces de integrar la simulación de un microcontrolador de una manera transparente para el programador, aún cuando se le restan las desventajas que esto presenta como la sobrecarga que aplica el intérprete del lenguaje multiplataforma a la computadora anfitriona.**

La presente investigación propone una solución basada en la ejecución de código manejado por una máquina virtual. En teoría con el uso de una maquina virtual los sistemas de simulación pueden ser reutilizados en diferentes plataformas computacionales sin necesidad de recompilar el código, siempre y cuando exista una implementación de la máquina virtual para el sistema operativo y arquitectura sobre la cual esta corriendo la computadora anfitriona.

***Palabras clave* — Microcontrolador, simulador, máquina virtual, biblioteca de software.**

I. INTRODUCCIÓN

En la electrónica actual, es indudable el impacto que representan los microcontroladores (MCU). Estos circuitos integrados a gran escala pueden ser herramientas útiles para la ingeniería siempre y cuando se encuentren en las manos de un ingeniero capaz de programarlos para un rendimiento óptimo de sus características. El desarrollo de aplicaciones con estos dispositivos ha dado origen a lo que se le se conoce como "sistemas embedded".

Manuscrito recibido el 1 de Diciembre de 2006. Este trabajo fue respaldado por el departamento de Ing. Eléctrica y Electrónica del Instituto Tecnológico de Sonora.

César Olea Aguilar es Ingeniero en Electrónica, con acentuación en Instrumentación y Control, egresado del Instituto Tecnológico de Sonora. Trabajó en la fábrica de software del ITSON (Novutek) y en la empresa CIMSA en Nogales Sonora. Actualmente cursa estudios de posgrado en el área de computación en el CICESE en Ensenada, B.C.N.

Eduardo Romero A. Autor hasta la fecha se ha de desempeñado como Profesor de Tiempo Completo del Instituto Tecnológico de Sonora en el Departamento de Ingeniería Eléctrica y Electrónica Instituto Tecnológico de Sonora; Ave. Antonio Caso S/N Col. Villa ITSON; Ciudad Obregón, Sonora, México; C.P. 85138; C.P. 85130; Tel: (644) 4109000, ext. 1200; Fax: (644) 4109001 (e-mail eromero@itson.mx).

La familia del M68HC11xx está formada por microcontroladores avanzados de 8 bits desarrollados con tecnología HCMOS. Cuentan con capacidades periféricas sofisticadas que se encuentran construidas en el mismo chip (*on-chip*). Operan con una velocidad nominal de 2 MHz y su diseño estático logra mantener el consumo de potencia bajo [1].

Los dispositivos de la familia HC11 se utilizan ampliamente en el ambiente educacional debido a su facilidad de programación y sus excelentes capacidades periféricas como convertidores analógico – digitales, puertos de entrada – salida de uso general, interfaz de comunicación serie y temporizadores. Todo lo anterior los hace un excelente candidato para usarlo como material de enseñanza en un curso avanzado de sistemas digitales.

También estos dispositivos son regularmente empleados en aplicaciones alimentadas por baterías o en la industria automotriz debido a sus modos de conservación de energía controlados por software [2].

En la actualidad, existen múltiples simuladores de la familia de microcontroladores HC11 y esto se debe a que fue muy popular en la época de los 90's y aún se utiliza por estudiantes, amateurs y profesionales de la electrónica para proyectos principalmente de robótica o instrumentación y control [3, 4].

La forma más común de simular el HC11 es interpretando el código máquina del programa (en formato S19), para después cargarlo a la memoria del simulador y comenzar el ciclo de búsqueda y decodificación modificando los registros internos y memoria como sea necesario [5].

El incentivo principal que motiva la creación de la biblioteca de simulación multiplataforma para el HC11 es tener disponible una plataforma para la creación de distintas herramientas que faciliten el aprendizaje y el desarrollo para el microcontrolador. Actualmente existen varios simuladores (algunos mostrados en la tabla I), pero ninguno que integre características que lo hagan especialmente atractivo al ambiente educativo y amateur.

Se eligió el lenguaje de programación Java para lograr máxima portabilidad en la biblioteca de programación con poco esfuerzo en este punto específico.

Java, desarrollado principalmente por James Gosling en Sun Microsystems, utiliza el paradigma de programación orientada a objetos. Desde el inicio fue concebido con cuatro principales puntos en mente: orientación a objetos, independencia de la plataforma, con bibliotecas de apoyo y facilidades en el lenguaje para redes y diseñado para ejecutar código de fuentes externas de manera segura.

TABLA I COMPARACIÓN DE DISTINTOS SIMULADORES PARA EL HC11

Nombre	Plataforma	Costo (dlls)	Aún activo
AVSim11	MS-DOS	Abandonado	No
THRSim11	Windows	$50	Si
Wookie	Windows	Gratis	No
6811wsim	Windows	Abandonado	No
EAF HC11 Simulator	Windows	$39.95 Estudiante	No

Gracias a los procesadores modernos y a las nuevas técnicas de simulación como la recompilación dinámica, el impacto en la velocidad de ejecución de simuladores en código manejado puede ser bastante eficiente manteniendo la multiplataforma.

II. ALCANCE DEL PROYECTO

La biblioteca de simulación se ha concentrado en las características básicas del HC11 como lo son el CPU y el mapa de memoria. La biblioteca no intenta replicar de manera exacta el comportamiento del HC11, aunque construye una infraestructura que puede servir para futuras ampliaciones en la programación de un simulador.

La simulación utiliza la técnica de interpretación. Esta técnica produce código lento pero es más fácil de mantener e implementar. La biblioteca de simulación es completamente independiente de cualquier interfaz gráfica, lo que quiere decir que con la biblioteca de simulación se pueden crear aplicaciones de escritorio con interfaz gráfica, aplicaciones de consola o aplicaciones Web.

Ninguno de los métodos de entrada/salida del microcontrolador fue simulado. Sin embargo, los registros que controlan a los puertos así como aquellos que reciben o externan el dato, si pueden ser simulados. Esto se debe a que en el HC11 se usa la E/S mapeada a memoria, con lo que, estos registros a final de cuentas son similares a los registros de propósito general.

La biblioteca de simulación soporta el juego de instrucciones completo del HC11, sin embargo algunas instrucciones no realizan ninguna operación. Al momento de cargar un programa a la memoria del dispositivo simulado, el cargador (*loader*) acepta cualquier instrucción válida en formato S19, sin embargo, las instrucciones pertinentes a sistemas no simulados no realizan ninguna acción en especial, tan solo dejan pasar los ciclos de reloj que se consumirían al ejecutar la instrucción

III. DESARROLLO DE LA BIBLIOTECA DE SIMULACIÓN

Al inicio del proceso de programación de la biblioteca fueron definidas dos clases que representan las dos estructuras de almacenamiento presentes en el HC11: *Jbyte* y *Jword* para representar bytes y palabras respectivamente. En la figura 1 se muestra la relación entre las clases.

Un *Jbyte* se encuentra definido como un arreglo de ocho posiciones de tipo booleano, en donde cada casilla del arreglo puede tener valor verdadero (1) o falso (0) como se muestra en la figura 2.

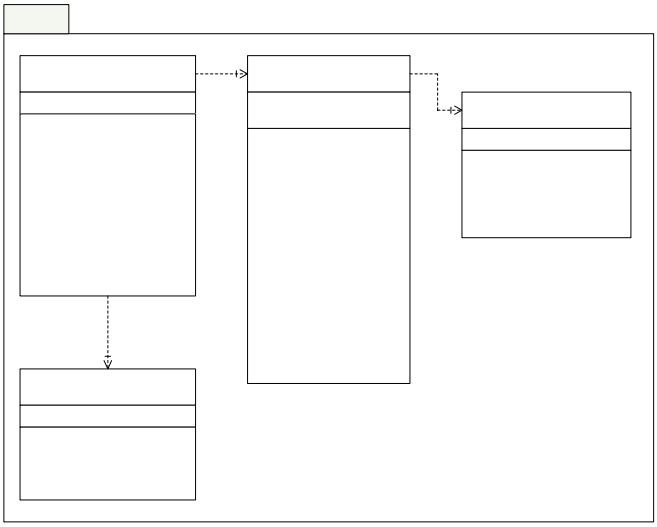

Fig. 1 Clases y la relación que existe entre ellas.

0	1	2	3	4	5	6	7
False	False	False	False	False	False	False	False

Fig. 2 Aalmacenamiento jbyte a 0.

La clase *Jword* esta formada internamente por dos *Jbyte,* representando la parte más significativa y menos significativa. Cada una de las clases se encarga de proporcionar métodos para escribir y leer un dato a la memoria que representan, así como también proporcionan una interfaz para realizar otro tipo de acciones como complementos, incrementos, conversiones entre representaciones numéricas, etc.

Para construir el mapa de memoria se creó una clase llamada *JmemoryRegister* que representa a una localidad de memoria. Su estructura se muestra en la figura 3.

De la clase *JmemoryRegister* heredan dos clases: RAM y ROM. Estas clases representan un byte del tipo designado por el nombre e implementan sus propios métodos de lectura y escritura, es decir, sobrescriben los métodos heredados de su clase padre.

Al momento de construir el mapa de memoria, se agruparon segmentos iguales de memoria y son estos grupos (representados por la clase *JmemoryObject*) los que se mapean. La clase *JmemoryObject* contiene un vector que guarda cualquier tipo de objetos hijo de la clase *JmemoryObject* (que hereden de *JmemoryObject*).

La clase *JmemoryObject* ya se asemeja más a un mapa de memoria completo. Para leer o escribir a estos grupos de memoria es necesario proporcionarle la dirección y el dato a escribir si es el caso. Los objetos *JmemoryObject* son concientes de los límites en los que se encuentran, para lo cual utilizan la dirección física en la que se encuentran mapeados y el tamaño del vector de memoria que contienen.

Cuando un *JmemoryObject* lee un dato exitosamente, este regresa como resultado un *JmemoryRegister* que contiene el dato leído. Este *JmemoryRegister* puede ser promovido a su clase original para no perder la implementación de sus métodos de lectura y escritura específicos.

Con el propósito de mantener un mapa de memoria coherente, se desarrollo la clase *JmemoryMap*. Esta clase encapsula al mapa de memoria del HC11 y se encuentra formada

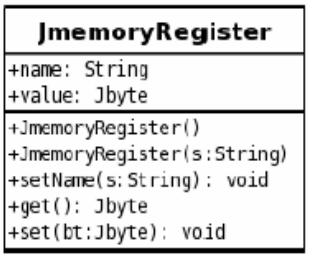

Fig. 3 Diagrama de clase para JmemoryRegister.

de varios objetos *JmemoryObject* los cuales, a su vez, contienen los bytes que forman al mapa de memoria.

Gracias a que las clases que forman el mapa de memoria manejan una abstracción de lo que es un byte mapeado en memoria (*JmemoryRegister*) se pueden crear distintos dispositivos que se incluyan en el mapa de memoria, manteniendo una interfaz común de lectura y escritura de datos, es decir, sin importar el tipo de memoria o dispositivo en el mapa, todos se leen y escriben llamando al mismo método y proporcionando los mismos datos y siempre se obtienen objetos del mismo tipo como resultado, pudiendo el usuario promoverlos a su tipo de clase original.

La clase HC11 encapsula todo el comportamiento del microcontrolador, este contiene un mapa de memoria y se encarga de controlar el ciclo de búsqueda y decodificación (*fetch/decode*) que a su vez se encarga de ejecutar por medio de reflexión a los métodos de la clase *Opcodes* la cual contiene todas las instrucciones que el HC11 puede ejecutar.

IV. USO DE LA BIBLIOTECA DE SIMULACIÓN

Para crear un microcontrolador HC11 simulado utilizando la biblioteca dentro de nuestro programa, basta con instanciar un objeto de tipo HC11.

```
HC11 micro = new HC11();
```

En este momento se tiene un objeto llamado micro el cual es un HC11 capaz de cargar a su memoria programas en formato S19, desensamblarlos y ejecutarlos paso a paso, con la opción de modificar el flujo del programa en cualquier momento. Cuando se instancia un HC11 nuevo, automáticamente se agregan tres bloques de memoria a su mapa: La RAM interna en la dirección 0x00, los registros especiales en la dirección 0x1000 y un vector de memoria RAM que comienza en 0xFF00.

Es posible agregar o eliminar bloques de memoria al mapa del microcontrolador. Continuando con el ejemplo del HC11 declarado anteriormente, si se quiere eliminar el bloque de memoria presente en 0xFF00.

```
micro.removeJmemoryObject(2);
```

El argumento del método *removeJmemoryObject* es debido a la posición del bloque de memoria en el microcontrolador. Como se mencionó anteriormente, el primer bloque (índice 0) lo ocupa la RAM interna, el segundo los registros y el tercero lo ocupa el bloque que se desea eliminar.

Si por el contrario, se desea agregar un bloque de memoria en la dirección 0x100 de tamaño 0x100 y del tipo RAM.

```
Jword dirInicio = new Jword(0x100);
Jword tam = new Jword(0x100);
RAM tipo = new RAM();
JmemoryObject miObjecto = new JmemoryObject(dirInicio, tam, tipo);
```

Primero se crea una palabra con la dirección de inicio, otra palabra con el tamaño del bloque y para finalizar un objeto que herede de *JmemoryRegister* (en este caso RAM) para especificar el tipo del bloque. Con esos datos se construye el nuevo objeto de memoria, sin embargo, aún hace falta mapearlo en el HC11.

```
micro.addJmemoryObject(miObjeto);
```

Con la sentencia anterior, el objeto de memoria es agregado al mapa del microcontrolador. Antes de agregarlo la clase *JmemoryMap* verifica que el objeto que se quiere agregar no viola ninguna de las reglas establecidas (no traslapes, no mayor a 0xFFFF) y si cumple, el objeto es agregado en la primera posición disponible.

Una vez que se tiene el objeto mapeado, puede ser leído y escrito de una manera que es común para todo lo mapeado en el microcontrolador.

```
micro.memoryMap.set(new Jword(0x100), new Jbyte(0xAA));
micro.memoryMap.get(new Jword(0x100));
```

La primera sentencia escribe en la dirección 0x100 el dato 0xAA. La segunda lee de la dirección 0x100 el dato contenido que por consecuencia será 0xAA.

La biblioteca de simulación contiene también un cargador de programa para el formato S19 así como un desensamblador. Para cargar un programa a la memoria del microcontrolador primero es necesario contar con los bloques de memoria necesarios para acomodar el programa. La clase responsable de cargar el programa se denomina *S19Loader* y cuenta con tan solo dos métodos: *loadS19* y *checksum*. El primero se encarga de cargar el programa mientras que el segundo tan solo hace una verificación del programa pero no lo carga a la memoria.

Para cargar un programa S19 primero se instancia un objeto de la clase *S19Loader*, para después llamar a su método *loadS19* e intentar cargar el programa leído a la memoria del microcontrolador.

```
String ruta = "C:\\progs\\hc11\\test1.s19";
S19Loader s19 = new S19Loader(ruta, micro);
S19.loadS19();
```

La ruta es dependiente del sistema operativo y por lo general se obtiene del usuario. En el momento que se llama *loadS19* sobre el objeto de tipo *S19Loader* se inicia la decodificación y carga del programa en formato S19.

Una vez que se tiene el programa en la memoria del microcontrolador, se debe modificar el contador de programa (PC) para que apunte a la primera dirección del programa cargado.

Suponiendo que el programa inicia en la dirección 0x100:

```
micro.setPC(0x100);
```

De la misma manera se puede alterar cualquiera de los registros internos del HC11, la tabla 2 lista los registros y los métodos utilizados para leer y escribir a ellos.

TABLA II. NOMBRE DE LOS REGISTROS INTERNOS.

Registro	Nombre de los métodos
PC	getPC, setPC
Acumulador A	getA, setA
Acumulador B	getB, setB
Acumulador D	getD, setD
Registro índice X	getX, setX
Registro índice Y	getY, setY
Apuntador de stack	getSP, setSP

Vale la pena señalar que estos registros internos no se acceden de la misma manera que los registros de memoria ya que estos no se encuentran mapeados en memoria, sino que se encuentran dentro de la CPU del HC11 [6].

La clase HC11 provee también métodos para controlar el estado del microcontrolador, como lo son las instrucciones de "reset" y "step". La primera pone a los registros especiales del HC11 en su estado inicial y la segunda se utiliza para ejecutar paso a paso el programa cargado en memoria.

```
micro.step();
```

En este momento el microcontrolador ejecuta el método *fetch* para traerse la instrucción de la dirección apuntada por PC y acto seguido ejecuta el método *decode*, encargado de ejecutar por medio de reflexión la instrucción correspondiente al código de operación traído de la memoria del microcontrolador.

Para descompilar el programa y mostrar el listado a partir de lo que se encuentra en la memoria del microcontrolador, la biblioteca utiliza una base de datos que necesita ser instalada en la computadora anfitriona. Al descompilador se le indica el dato a descompilar y la dirección en la cual se encuentra. La dirección se utiliza para mostrar los desplazamientos o la dirección de los datos que se van a ocupar como argumento de la instrucción.

Al recibir la instrucción en hexadecimal, la biblioteca ejecuta una sentencia SQL sobre la base de datos que contiene las instrucciones. La sentencia ejecutada es la siguiente:

```
SELECT * FROM opcodes WHERE OPID = + 'opcode en hex'.
```

Lo que se traduce a: "selecciona todos aquellos renglones de la base de datos en los cuales el campo OPID coincida con el opcode enviado". No pueden existir códigos de operación repetidos por lo que no puede darse el caso en el que se obtengan más de un renglón. Como resultado de la consulta, el método regresa una cadena con la instrucción decompilada.

Todas las funciones de la biblioteca de simulación funcionan de la misma manera y están diseñadas con la facilidad de uso en mente. De cualquier manera, es necesario consultar la documentación de las clases e inclusive ver el código fuente para entender mejor el funcionamiento de cada una de las clases y conocer las posibilidades de uso.

V. CONCLUSIONES

La biblioteca de simulación producto de esta investigación no es un producto terminado todavía y aún le hacen falta piezas clave para elevar su utilidad en cualquier ambiente en el que se pretenda utilizar. Algunas de las piezas clave faltantes son:

Creación de los registros mapeados a memoria.

Implementación correcta de inicialización de memoria ROM con un valor por defecto.

Implementación de los diferentes modos del microcontrolador (expandido, "bootstrap", "special test").

Interrupciones.

Es importante estar familiarizado con el lenguaje de programación Java y con los conceptos de la programación orientada a objetos, para poder comprender la relación que existe entre las diferentes clases y hacer uso correcto de la biblioteca.

Con el desarrollo y utilización de esta biblioteca de simulación se puede comprobar que el sobrepeso que ejerce el código manejado por una máquina virtual es prácticamente nulo para la simulación de dispositivos con velocidades de reloj relativamente bajas, como lo es el HC11. Es por lo mismo que una aproximación de interpretación para simular al HC11 es más que adecuada, siendo la recompilación dinámica demasiado complicada sin grandes beneficios para este caso.

Con esta biblioteca se pueden desarrollar herramientas que faciliten el aprendizaje y enseñanza del microcontrolador HC11 como lo son: aplicaciones del lado del servidor (web), un simulador gráfico específicamente diseñado para las necesidades de la institución, una herramienta de búsqueda de códigos de operaciones con ejemplos "vivos" en la que se pueda comprobar el funcionamiento de una operación al mismo tiempo que se tiene la definición y explicación de la instrucción, y cualquier otra herramienta útil que se imagine.

REFERENCIAS

[1] Motorola Inc., *M68HC11, The Referente Manual,* USA, 1991.
[2] Motorola Inc., M68HC11A8, *Data Technical Manual*, USA, 1991.
[3] Driscoll, F., *Data Acquisition and Process Control with the M68HC11 Microcontroller*, USA, Macmillan Publishing Company, 1994.
[4] Huang, Han-Way, *MC68HC11 an Introduction: Software and Hardware Interfacing*, USA, Thomson Learning, 1996.
[5] Motorola Inc., *Freeware 8-bit Cross Assemblers User's Manual,* USA, Kevin Anderson Field Applications Engineer.
[6] Spasov, Peter, *The Microcontroller Technology*, USA, Prentice Hall.

César Olea Aguilar es Ingeniero en Electrónica, con acentuación en Instrumentación y Control, egresado del Instituto Tecnológico de Sonora.
Sus áreas de interés son: Programación en C/C++, Java, ensamblador para el x86, ensamblador para el microcontrolador HC11, y los sistemas digitales en general.
Trabajó en la fábrica de software del ITSON (Novutek) y para la empresa CIMSA en Nogales Sonora. Actualmente cursa estudios de posgrado en el área de computación en el CICESE, B:C:N.
Es activista del movimiento de software libre y actualmente se encuentra desarrollando el simulador Jiss para el HC11 utilizando la biblioteca de simulación que creó como proyecto de titulación. Promueve los sistemas operativos Open Source y su uso tanto en empresas particulares como en instituciones públicas.

Eduardo Romero Aguirre obtuvo el grado de Ingeniero en Electrónica opción Instrumentación en el Instituto Tecnológico de Orizaba en 1995 y el grado de Maestro en Ciencias en Ingeniería Electrónica en el área de Sistemas Digitales, en el Centro Nacional de Investigación y Desarrollo Tecnológico – CENIDET de Cuernavaca, Morelos, México en 1999. Ha realizado diversos proyectos relacionados con sistemas de adquisición de datos. Actualmente labora como profesor/investigador de tiempo completo en el Instituto Tecnológico de Sonora.