CARGADOR DE APLICACIONES PARA EL MICROCONTROLADOR ATMEGA328P BUSCANDO APROVECHAR LOS RECURSOS DE UNA TARJETA ARDUINO

Felipe Santiago Espinosa

Universidad Tecnológica de la Mixteca, Instituto de Electrónica y Mecatrónica

fsantiag@mixteco.utm.mx

Zenón Belarmino Martínez Cruz

Universidad Tecnológica de la Mixteca, Instituto de Electrónica y Mecatrónica

zenon.mn.23@gmail.com

Resumen

El desarrollo de aplicaciones con microcontroladores requiere de la continua programación del dispositivo. Generalmente el MCU se retira del sistema para ser grabado aunque también es posible su programación *"in system"*, con lo que se ahorra tiempo y evita el daño de pines. Con Arduino se simplificó la curva de desarrollo porque facilita la programación desde su IDE y cuenta con numerosas bibliotecas para el manejo de periféricos. La tarjeta Arduino es soportada por un AVR y estos MCU tienen dos secciones en su memoria de código: arranque y aplicación. El MCU de Arduino tiene un cargador que facilita la programación de aplicaciones. Sin embargo, desde Arduino no se realiza un acceso a registros de manera que los recursos suelen usarse ineficientemente y se complica la ejecución concurrente de tareas que se consigue mediante interrupciones. Los desarrolladores experimentados prefieren usar un microcontrolador independiente. Buscando aprovechar el hardware de Arduino, se desarrolló un cargador para un ATMega328P, un programa que desde la sección de arranque facilita el grabado de la sección de aplicación sin requerir un programador adicional. Con ello, es

posible trabajar en ensamblador o C y realizar las descargas en el MCU por medio de una aplicación realizada en Java, la cual complementa al cargador.
Palabra(s) Clave(s): Arduino, aplicación en Java, auto-programación, cargador.

1. Introducción

Un microcontrolador o MCU (*Micro-Controller Unit*) es un Circuito Integrado VLSI que contiene una CPU, memoria para código, memoria para datos, temporizadores, fuentes de interrupción y otros recursos necesarios para el desarrollo de aplicaciones. Un MCU incluye los elementos necesarios para ser considerado como una computadora en un chip pero frecuentemente no es tratado como tal, ya que su uso típico consiste en el desempeño de funciones de "control" interactuando con el "mundo real" para monitorear condiciones (a través de sensores) y en respuesta a ello, encender o apagar dispositivos (por medio de actuadores) [1].

Los microcontroladores están enfocados a aplicaciones de propósito específico, como electrodomésticos, controles remotos, equipo de audio, juguetes, etc., en donde se deben cubrir requerimientos de costo, consumo de potencia y espacio más que una demanda de cómputo exhaustivo [2]. Las características de los MCU los hacen ideales para sistemas embebidos autónomos, así como para sistemas a medida que son parte de un proceso de automatización; por lo que su estudio es fundamental durante la formación de profesionistas en las áreas de electrónica, mecatrónica o afines.

Cuando se desarrollan sistemas con base en un MCU, en algún punto del proceso se debe programar al dispositivo, esto implica retirarlo del sistema para llevarlo a un grabador o bien, conectar el grabador en algún puerto del sistema y realizar la descarga del nuevo programa (programación *in system*). En ambos casos se requiere de un hardware de programación externo al sistema.

Para simplificar el desarrollo de sistemas con microcontroladores, en el Instituto de Diseño e Interacción IVREA se creó a Arduino, una herramienta simple de prototipado rápido, dirigida a estudiantes sin fundamentos en electrónica y programación. Arduino es una plataforma *open-source* basado en hardware y

software de fácil uso, las tarjetas pueden leer entradas (sensores, botones, texto, etc.) y generar salidas (activando un motor, encendiendo un LED, publicando mensajes, etc.) [3]. Las tareas se programan con el envío de instrucciones, desde el IDE de Arduino se realiza la codificación y descarga en el microcontrolador de la tarjeta, sin requerir hardware adicional.

Actualmente Arduino es usado por una comunidad mucho más amplia, sus tarjetas han evolucionado para adaptarse a nuevas necesidades y proyectos, los *shields* o módulos de expansión se conectan a la tarjeta base para ampliar sus capacidades. Comercialmente se encuentran disponibles *shields* para el manejo de motores, lectura de sensores, comunicaciones con diferentes protocolos, interfaces alfanuméricas y gráficas, etc. En muchos casos se mantiene la misma filosofía: deben ser fáciles de montar y de bajo costo.

Arduino es muy conveniente para el desarrollador con poca experiencia porque puede realizar proyectos en poco tiempo sin estudiar la arquitectura del microcontrolador; además, por el extenso número de bibliotecas para periféricos, es posible emplear *shields* desconociendo la forma en cómo operan los dispositivos incluidos [4].

Las tarjetas están basadas en microcontroladores AVR y estos dispositivos tienen una variedad de recursos, por ejemplo, las versiones Arduino Nano, Micro, UNO y LilyPad incluyen un ATMega328P, este dispositivo tiene los recursos que se resumen en la tabla 1 [5]. La memoria de código se divide en dos secciones, una sección de arranque y otra de aplicación. Arduino hace uso de esta característica para dejar un programa residente en la sección de arranque, mediante el que se realiza la descarga de programas que se ubicarán en la sección de aplicación.

En la etapa inicial de la formación de un ingeniero electrónico o mecatrónico es válido y conveniente el uso de herramientas como Arduino; sin embargo, en la medida en que su formación avanza, es necesario el conocimiento de la organización de los MCU para que realice sistemas que exijan el máximo desempeño del chip. El desarrollador debe tener la libertad de usar las interrupciones generadas por los recursos internos para que haga dos o más

tareas en forma concurrente y realice sistemas embebidos con operación en tiempo real.

Tabla 1 Características del microcontrolador ATMega328.

Recurso	Disponibilidad	Recurso	Disponibilidad
Flash (código)	32 kB	Interfaz TWI	1
EEPROM (datos)	1 kB	ADC de 10 bits	1 con 6 canales
SRAM (datos)	2 kB	Comparador Analógico	1
Terminales I/O	23	Watchdog Timer	1
Frec. Máxima	20 MHz	Oscilador interno	1 MHz/8 MHz
Voltaje de Operación	1.8 – 5.5 V	Multiplicador por HW	Si
Temporizador (16 bits)	1	Interrupciones	26
Temporizador (8 bits)	2	Int. Externas	2
Canales PWM	6	Int. Por cambio en pines	3
Reloj de tiempo real	1	Programación *in system*	Si
USART	1	Auto programación	Si
SPI Maestro/Esclavo	1	Interfaz *Debug Wire*	Si

Esto implica programar en un nivel más bajo al propuesto por Arduino y emplear lenguaje ensamblador o C, con ambas opciones se realiza un acceso directo a los registros I/O, mediante los cuales se realiza la configuración, control y estado de los recursos internos [1]. El máximo aprovechamiento de un MCU se obtiene con la metodología tradicional en la que se emplea solo al microcontrolador, se codifica y simula antes de descargar, para después programar al MCU, integrar y evaluar al sistema.

Con la finalidad de aprovechar las ventajas de los dos paradigmas, se presenta la implementación de un cargador AVR, un pequeño programa que reside en la sección de arranque del microcontrolador y remplaza al cargador de arduino. La presencia del cargador facilita la auto-programación del MCU y se auxilia de una aplicación en Java desde donde se selecciona el archivo a descargar. El cargador AVR no introduce código adicional a la aplicación del usuario, el sistema tendrá el mismo desempeño que con el MCU solo. Similar al paradigma tradicional, el desarrollador codifica en C o ensamblador, compila y simula. Una vez generado el código máquina (archivo *HEX*), la aplicación en Java transfiere el código para su grabado en el MCU.

La operación del cargador AVR inicia cuando el sistema se energiza o después de un reset, el cargador espera 15 segundos por una interrupción serial para la descarga de una nueva aplicación, concluida la carga o terminado el periodo, se realiza un salto a la sección de aplicación. Las tarjetas arduino incluyen un adaptador USB-USART, este es aprovechado por el cargador AVR, por lo que no se requiere hardware adicional para la programación de nuevas aplicaciones.

El cargador AVR no es exclusivo para las tarjetas arduino, pero se enfocó a un ATMega328 para aprovechar el hardware de una tarjeta Arduino UNO con los *shields* que para ella se han desarrollado, maximizando el rendimiento al desarrollar en lenguaje C o ensamblador. Un trabajo similar se encuentra disponible en el sitio ATmega32-AVR [6].

2. Desarrollo

Para este trabajo se realizaron dos programas, el cargador que se ubica en la sección de arranque del microcontrolador y el programa en Java que facilita la descarga de aplicaciones. Ambos programas son descritos en las siguientes secciones.

El cargador de arranque en el ATMega328

El cargador proporciona un mecanismo de auto-programación para actualizar el código desde el mismo microcontrolador. Esta característica permite actualizaciones flexibles del software de aplicación usando un programa residente en la memoria flash. En esta sección se describen las características de la memoria flash y los bits de configuración y seguridad (fusibles) que deben modificarse para la adecuada operación del cargador, la configuración de los fusibles se realiza durante la programación del cargador.

La memoria flash del ATMega328

La memoria flash del ATMega328 es de 16 K de palabras de 16 bits, se estructura en páginas de 64 palabras teniendo un total de 256 páginas. El acceso

para el borrado y escritura implica la modificación de una página completa. La lectura es diferente, ésta puede realizarse con un acceso por bytes.

Como previamente se ha citado, la memoria tiene las secciones de arranque y aplicación, el número de páginas para cada sección se determina con los fusibles **BOOTSZ**. En la tabla 2 se exhiben los diferentes tamaños posibles, así como el intervalo de direcciones en cada combinación, una aplicación puede tener acceso al código escrito en la sección de arranque, por medio de llamadas a rutinas o saltos. Los dispositivos son comercializados con **BOOTSZ** = "00", dejando 32 páginas para la sección de arranque.

Tabla 2 Diferentes configuraciones para BOOTSZ en un ATMega328. [1:0]

BOOTSZ 1	BOOTSZ 0	Aplicación	Arranque	Tamaño (Arranque)	Páginas
1	1	0x0000 – 0x3EFF	0x3F00 – 0x3FFF	256 words	4
1	0	0x0000 – 0x3DFF	0x3E00 – 0x3FFF	512 words	8
0	1	0x0000 – 0x3BFF	0x3C00 – 0x3FFF	1024 words	16
0	0	0x0000 – 0x37FF	0x3800 – 0x3FFF	2048 words	32

La sección de aplicación inicia en la dirección 0 por lo que al energizar al MCU comienza con la ejecución de la aplicación. No obstante, con el fusible **BOOTRST** se consigue que después de un *reset* la CPU inicie su ejecución en la sección de arranque y no en la dirección 0.

Los vectores de interrupción se ubican en las primeras direcciones de la memoria flash por lo que están en la sección de aplicación; sin embargo, modificando al registro **MCUCR** (*MCU Control Register*) se pueden desplazar los vectores de interrupción a la sección de aplicación. Los bits de interés son **IVSEL** (*Interrupt Vector Select*) e **IVCE** (*Interrupt Vector Change Enable*), ocupando las posiciones 1 y 0 en el registro **MCUCR**.

Con un 0 en **IVSEL** los vectores de interrupción se ubican al inicio de la memoria flash y con un 1 se desplazan al inicio de la sección de arranque, cuya dirección depende de los fusibles **BOOTSZ** (tabla 2). El bit **IVCE** es un habilitador que evita cambios no deseados en la ubicación de los vectores de interrupción. Cualquier ajuste requiere la puesta en alto de **IVCE** y dentro de los 4 ciclos de reloj

siguientes se debe escribir el valor deseado en **IVSEL**. La programación de **BOOTRST** es independiente al estado del **IVSEL**, un programa puede iniciar su ejecución en la sección de arranque, conservando los vectores de interrupciones al inicio de la memoria flash. Los vectores de interrupción se pueden mover en tiempo de ejecución.

Comportamiento del cargador

El MCU inicia ejecutando al cargador (ubicado en la sección de arranque), el cual mueve los vectores de interrupción y con el apoyo del temporizador 1 espera durante 15 segundos por un posible comando desde una PC vía puerto serie. Durante ese periodo el MCU puede recibir y atender comandos para el borrado del dispositivo, la lectura de la aplicación actual o la escritura de una nueva aplicación; terminado el periodo sin más comandos, el cargador regresa el vector de interrupciones a su ubicación original, los registros modificados a su valor original y bifurca la ejecución a la dirección 0 para iniciar con el programa de aplicación. En la figura 1 se muestra el flujo que sigue el cargador de arranque.

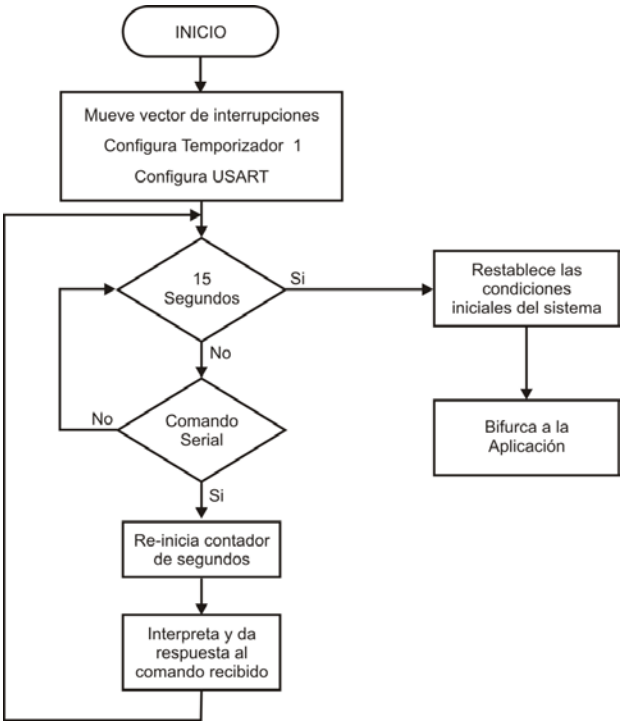

Figura 1 Diagrama de flujo del Cargador AVR.

Restricciones de acceso a la memoria flash

El acceso a la memoria flash se realiza con instrucciones de carga (**LMP,** *Load Program Memory*) y almacenamiento (**SPM***, Store Program Memory*). Los MCU se comercializan sin restricciones en el uso de estas instrucciones, es decir, el contenido de las dos secciones se puede modificar con secuencias de código ubicadas en cualquier sección. Sin embargo, con los fusibles denominados Bits de Bloqueo de Arranque (**BLB**, *Boot Lock Bits*) es posible configurar un nivel de protección en cada sección. Con ellos se determina si la memoria flash queda protegida de una actualización, si sólo la sección de arranque queda protegida, si sólo la sección de aplicación queda protegida o ambas secciones quedan sin protección. Para el cargador AVR se mantiene la configuración de fábrica (sin restricciones), de manera que las aplicaciones pueden modificar toda la memoria de código, incluyendo el espacio ocupado por el cargador.

Capacidades para Leer Mientras Escribe

Además de la división entre la sección de aplicación y la sección de arranque, la memoria flash también se divide en dos secciones de tamaño fijo, la primera es una sección con capacidad de Leer Mientras Escribe (RWW, *Read While Write*) y la segunda sección restringe el acceso, de manera que No se puede Leer Mientras Escribe (NRWW, *No Read While Write*). En la tabla 3 se muestra el tamaño de cada sección en un ATMega328, debe notarse que la sección NRWW abarca el espacio que originalmente tiene la sección de arranque.

Tabla 3 Límites de la sección RWW.

Sección	Páginas	Dirección
Leer mientras escribe (RWW)	224	0x0000 – 0x37FF
No leer mientras escribe (NRWW)	32	0x3800 – 0x3FFF

La diferencia fundamental entre la sección RWW y la NRWW es que:

- Cuando se está borrando o escribiendo una página localizada en la sección RWW, la sección NRWW puede ser leída durante esta operación.

- Cuando se está borrando o escribiendo una página ubicada en la sección NRWW, la CPU se detiene hasta que concluya la operación.

El cargador se ubica en la sección NRWW y las aplicaciones quedarán en la sección RWW, por lo que la CPU no se detendrá cuando el cargador lea o escriba una aplicación. Leer Mientras Escribe significa lecturas de la sección NRWW relacionadas con escrituras en RWW. En la figura 2 se muestran las secciones y el uso del apuntador Z para referenciar la ubicación de la página a modificar.

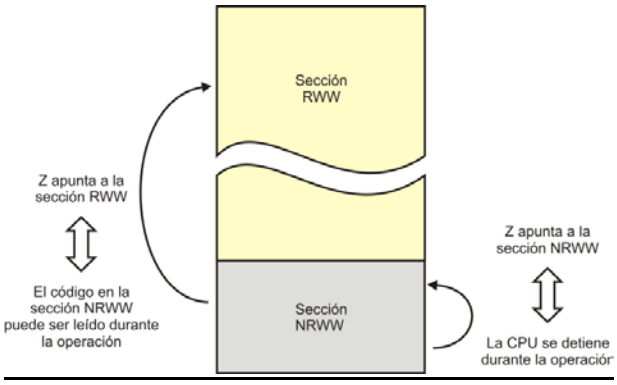

Figura 2 Acceso con el apuntador Z a las secciones RWW y NRWW.

Escritura y borrado en la memoria flash

El repertorio de instrucciones de los AVR incluye a la instrucción **SPM**, con ella se escribe la palabra de 16 bits ubicada en los registros R1:R0 en la dirección de memoria flash referida con el registro Z. Sin embargo, su uso no es directo porque el acceso para la escritura o borrado se realiza por páginas completas.

El hardware incluye un buffer de memoria del tamaño de una página, en el que se hace un almacenamiento temporal antes de escribir en la memoria Flash. El espacio del buffer es independiente de la memoria SRAM, es sólo de escritura y debe ser llenado palabra por palabra. La escritura en el buffer se auxilia del registro **SPMCSR** (*Store Program Memory Control and Status Register*) porque su bit menos significativo es el habilitador de escritura (**SPMEN**, *Store Program Memory Enable*). El procedimiento para escribir una palabra en el buffer es:

- Colocar la palabra en R1:R0.
- Apuntar a la dirección de escritura con el registro Z.

- Poner el alto al bit **SPMEN** del registro **SPMCSR**.
- Dentro de los 4 ciclos de reloj siguientes, ejecutar la instrucción SPM.

El registro **SPMCSR** tiene otros 7 bits relacionados con el acceso a la memoria flash (tabla 4), de los cuales, sólo 3 son empleados por el cargador AVR, su uso se describe a continuación:

- Bit 4 – **RWWSRE**: Habilita la lectura de la sección RWW. El acceso a la sección RWW se bloquea después del borrado o escritura de una página. Para rehabilitar su acceso, el bit **RWWSRE** se debe poner en alto junto con el bit **SPMEN** y ejecutar la instrucción **SPM** dentro de los 4 ciclos de reloj siguientes.
- Bit 2 – **PGWRT**: Habilita la escritura de una página.Una vez que el buffer está lleno, con este bit en alto junto con el bit **SPMEN**, al ejecutar la instrucción **SPM** dentro de los 4 ciclos de reloj siguientes se inicia con la escritura de la página. En la parte alta del apuntador Z se debe colocar el número de página de la memoria flash y el valor de los registros R1:R0 es ignorado.
- Bit 1 – **PGERS**: Habilita el borrado de una página.Para el borrado de una página se deben poner en alto los bits **PGERS** y **SPMEN,** y durante los 4 ciclos de reloj siguientes se debe ejecutar la instrucción **SPM**. La página debe ser direccionada con la parte alta del apuntador Z y el valor de los registros R1:R0 es ignorado.

Tabla 4 Registro SPMCSR.

7	6	5	4	3	2	1	0
SPMIE	RWWSB	SIGRD	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN

Las operaciones de escritura en el buffer, rehabilitación de acceso, escritura y borrado de página fallan si la instrucción **SPM** no se ejecuta dentro de los 4 ciclos de reloj siguientes a la puesta en alto de los bits correspondientes. El bit **SPMEN** se pone en bajo automáticamente al terminar cualquier operación.

Direccionamiento de la memoria flash

El contador de programa (PC) es el registro encargado de direccionar la memoria de código, para el mapa completo de un ATMega328 se requieren de 14 bits (2^14 = 16K). Los bits del PC se dividen en 2 partes, con los 8 bits más significativos (PC[13:6]) se selecciona una página (PCPAGE, son 256 páginas) y con los 6 bits menos significativos (PC[5:0]) una palabra dentro de la página (PCWORD, cada página tiene 64 palabras). En las operaciones que usan la instrucción **SMP,** el apuntador Z debe hacer referencia a la memoria flash con la misma distribución de bits que el PC pero ignorando al bit menos significativo de Z. En la figura 3 se muestra la distribución de los bits para direccionar la memoria flash, para la escritura en el buffer se utilizan Z[6:1] y para la escritura y borrado de páginas se usan Z[14:7].

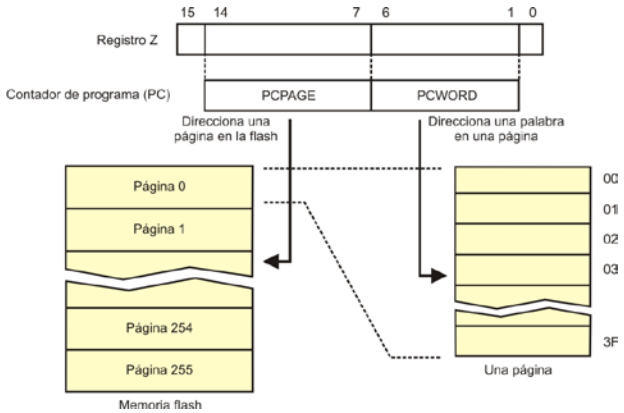

Figura 3 Distribución de bits para direccionar la memoria de código.

Las operaciones de escritura y borrado se realizan en varios ciclos de reloj, esto requiere la inserción de un *latch* intermedio en el que se captura la dirección proporcionada por el apuntador Z, de manera que Z puede ser utilizado para otras tareas después de haber iniciado la operación.

Consideraciones para la codificación y descarga del cargador

El cargador AVR tiene el comportamiento mostrado en la figura 1, sin embargo, el temporizador 1 y la USART son manejados por interrupción, el vector de interrupciones se mueve al iniciar con la ejecución del cargador y regresa a su posición original al dar paso a la ejecución de la aplicación.

La codificación se realizó en lenguaje ensamblador porque se simplifica su ubicación en la memoria flash, básicamente se utiliza la directiva **org**. Además, se conoce con certeza qué registros han sido modificados, registros que deben regresar a su valor de *reset* cuando se de paso a la aplicación.

El cargador AVR es evaluado en una tarjeta Arduino UNO y ésta tiene un cristal externo, de manera que las terminales PB6 y PB7 (XTAL1 y XTAL2) no pueden ser usadas con un propósito general, de antemano no tienen presencia en los conectores disponibles para el usuario. El manual de Arduino especifica un cristal de 16 MHz, no obstante, hay implementaciones comerciales con cristales diferentes. En el cargador se prefirió utilizar el oscilador interno para que pueda funcionar sin importar la frecuencia del cristal incluido en la tarjeta.

El ATMega328 tiene un oscilador interno calibrado a una frecuencia de 8 MHz y por medio del fusible **CLKDIV8** se realiza una división entre 8, los dispositivos se comercializan con este fusible programado operando a 1 MHz. En el cargador se desprogramó para contar con una velocidad mayor de operación, de manera que 8 MHz es la frecuencia de la tarjeta y con ella se calcula la velocidad de comunicación serial que es de 9600 bps.

Los fusibles a modificar en el momento de realizar la descarga son: **CLKDIV8** para quitar el factor de división al oscilador interno y establecer como reloj del sistema al oscilador interno de 8 MHz y **BOOTRST** para definir que el MCU iniciará su ejecución en la sección de arranque y no en la de aplicación.

No se cambia el tamaño de la sección de arranque, se mantiene el tamaño original de 32 páginas porque corresponde con el tamaño de la sección **NRWW**, de esta manera una aplicación sólo ocupará el espacio **RWW** y no se detendrá la ejecución del MCU mientras se actualiza. Esto limita el tamaño de las aplicaciones a un espacio máximo de 14 K x 16 bits.

Comunicación entre la aplicación Java y el MCU

La comunicación con el cargador se realiza mediante paquetes de datos que se envían al MCU desde una PC por medio del puerto serie. Cada paquete incluye un byte con el comando y la información necesaria para su ejecución. El MCU no es

activo por sí solo, sólo responde a los paquetes que la PC le envía bajo un protocolo protegido con una redundancia cíclica de 16 bits [7]. Se manejan 5 paquetes, inician con el byte 0xAA, seguidos por el byte de comando, argumentos si se requieren y terminan con el CRC.

Los paquetes de datos para la interacción PC-MCU son:

- Reconocimiento: Este paquete sirve para validar la conexión o para respuestas exitosas. La PC envía el comando 0xAA y si el MCU está conectado y ejecutando al cargador, responderá con la misma señalización. El paquete completo es:

  0xAA 0xAA CRC-16 (2 bytes)

- Sin reconocimiento: El comando es una respuesta del MCU, utiliza al byte 0x11 y significa que el MCU no conoce el comando recibido o que ocurrió un error en el envío del paquete, reflejado en el cálculo del CRC. El paquete completo es:

  0xAA 0x11 CRC-16 (2 bytes)

  Si la PC recibe esta respuesta debe reenviar nuevamente al paquete para su correcta ejecución.

- Escritura: El comando es el byte 0x1A, con este comando la PC solicita la escritura de una página de código en la memoria flash del MCU, el paquete incluye la dirección de escritura y los 128 bytes de código (1 página tiene 64 palabras de 16 bits). El paquete tiene los campos:

  0xAA 0x1A Dirección (2 bytes) Datos (128 bytes) CRC-16 (2bytes)

- El MCU responde con un paquete de reconocimiento (0xAA) si la escritura se hizo correctamente o sin reconocimiento (0x11) en caso contrario. Una aplicación puede contener varias páginas, de manera que el comando se puede ejecutar en repetidas ocasiones cuando se solicite la grabación de una nueva aplicación.

  Lectura: El comando se distingue con el byte 0xA3, con este comando la PC solicita una página al MCU, para ello le envía la dirección de lectura, el paquete completo es:

  0xAA 0xA3 Dirección (2 bytes) CRC-16 (2bytes)

Con este comando el MCU no responde con el paquete de reconocimiento, sino con la información solicitada. El MCU envía el paquete:

0xAA 0xA3 Datos (128 bytes) CRC-16 (2bytes)

- Borrado: El comando se identifica con el byte 0xB1 y desde la PC se solicita el borrado de la sección de aplicación (224 páginas, 28 Kbytes). El paquete es:

0xAA 0xB1 CRC-16 (2bytes)

Aplicación en Java

El programa se implementa con ayuda del entorno de desarrollo Netbeans IDE 8.0, para obtener una interfaz amigable con el usuario. El proyecto lleva por nombre BL y cuenta con dos clases dedicadas a la comunicación serial, específicamente son dos hilos: Hilo_Conector e Hilo_Receptor que implementan a Runnable de modo que ambos subprocesos se pueden ejecutar simultáneamente [7]. La clase principal implementa un jFrame (ventana principal), en donde se insertan los botones, campos de texto y etiquetas necesarios para elegir el puerto serial a utilizar, ejecutar las acciones de Iniciar, Abrir archivo, Escribir, Leer, Borrar y visualizar el archivo HEX en una ventana secundaria.

Al ejecutar la aplicación se crean los objetos conector y receptor derivados de las clases Hilo_Conector e Hilo_Receptor respectivamente. Dentro de las clases se sobrescribe el método Run() que contiene el código que se ejecutará en un ciclo infinito. Ambas clases utilizan las funciones incluidas en la biblioteca Giovynet Driver. Las funciones de Giovynet Driver permiten el uso del lenguaje Java para crear aplicaciones que requieran implementar la comunicación serial con dispositivos externos, las funciones principales que se utilizan son: reciveSingleCharAsInteger() y sendSingleData() [8]. El primer hilo verifica que existan puertos seriales relacionados a la computadora y los inserta como nuevos elementos al comboBox de la interfaz principal. El segundo hilo realiza la lectura del bus de comunicaciones, con el fin de verificar que exista una conexión con el microcontrolador y obtener su respuesta a cada uno de los comandos recibidos.

El botón **Iniciar** configura todos los parámetros de la comunicación serial que luego serán parte del argumento del constructor de la clase Com. El botón **Abrir**

Archivo utiliza un filtro para visualizar únicamente archivos con extensión HEX, incluye una función que permite convertir los caracteres en ASCII a su valor entero para ser presentados en una ventana secundaria, los datos leídos se almacenan en un ArrayList. El botón **Escribir** almacena la trama con los comandos, las direcciones y el CRC-16 correspondiente a cada una de las páginas que conforman al archivo HEX.

Con ayuda de la clase jOptionPane se crean cuadros de diálogo que permiten al usuario identificar el estado de la comunicación serial, como en el caso de una escritura exitosa, un error en el borrado o un problema en la conexión. Utilizando únicamente una línea se pueden implementar estos cuadros de diálogo.

3. Resultados

En la figura 4 se observa el resultado final de la interfaz que permite la interacción del usuario y la tarjeta Arduino. Automáticamente la aplicación detecta los puertos disponibles en la computadora y permite al usuario elegir el correcto para después presionar el botón **Iniciar** y configurar la comunicación serial. Los botones de **Escribir**, **Leer** y **Borrar** se encuentran desactivados al inicio de la aplicación.

Figura 4 Ventana principal de la aplicación en Java.

El usuario tiene la posibilidad de verificar si la conexión entre la computadora y la tarjeta existe dando clic en *Herramientas/Comprobar conexión*, como se puede

observar en la figura 5. Por medio de un cuadro de diálogo se reconoce el estado de la conexión, ya sea conectado o desconectado. Los botones en la parte inferior de la interfaz corresponden a las acciones de: escribir archivo, leer y borrar, que se habilitan al comprobar la conexión.

Figura 5 Comprobación de la conexión computadora-tarjeta.

Con un programa sencillo se realizaron las pruebas de Escribir y Borrar. En la figura 6 se observa el microcontrolador funcionando después de realizar la escritura de un programa que enciende y apaga un par de leds, en la pantalla de la computadora aparece un cuadro de diálogo indicando que la escritura se realizó con éxito.

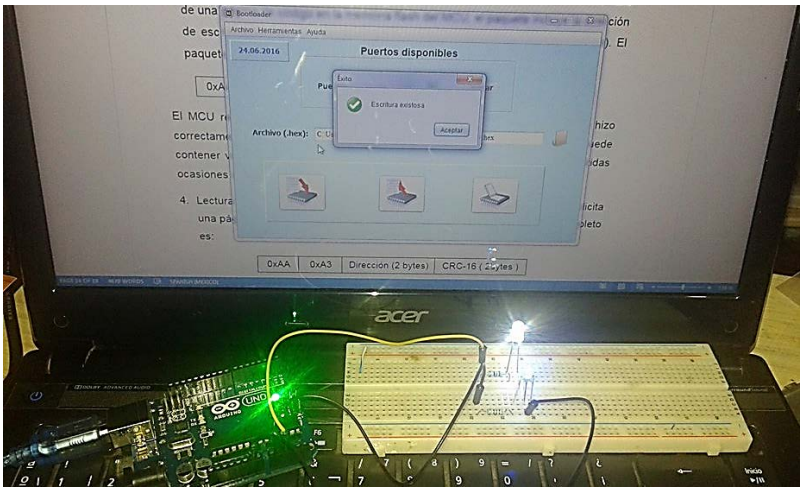

Figura 6 Aplicación en Java y el microcontrolador ATMega328 en operación.

4. Discusión

Entre los desarrolladores de aplicaciones con microcontroladores ha habido una extensa discusión relacionada con el lenguaje de programación empleado, están quienes defienden de manera tajante el uso de ensamblador argumentando un mejor aprovechamiento de recursos y por otro lado, están quienes prefieren los lenguajes de alto nivel porque es más fácil estructurar los programas y por lo tanto, se reduce el tiempo de desarrollo.

Algo similar ocurre entre el uso de Arduino y un microcontrolador independiente, con Arduino se reduce el tiempo de desarrollo y se pueden hacer aplicaciones rápidas aun desconociendo la organización del microcontrolador, la contraparte es que los recursos internos del MCU no se aprovechan al máximo y se imposibilitan algunas tareas que requieren respuestas inmediatas. De manera que, de acuerdo con la experiencia del desarrollador, pueden surgir diversos argumentos de por qué una propuesta es mejor que otra.

Con el desarrollo del cargador AVR se intentan aprovechar las ventajas de ambas propuestas, dado que permite el uso directo del microcontrolador AVR disponible en una tarjeta Arduino.

5. Conclusiones

La metodología tradicional del desarrollo de sistemas basados en microcontroladores se ve muy favorecida con la inclusión del cargador AVR en el MCU porque se simplifica significativamente el proceso de desarrollo al realizar la descarga de nuevas aplicaciones como si se tratase de un conjunto de datos.

El hecho de emplear una tarjeta Arduino como destino del cargador también proporciona grandes ventajas, dado que el desarrollador no invierte tiempo en el desarrollo de hardware y puede disponer de los diferentes *shields* existentes en el mercado.

El cargador AVR y la aplicación en Java complementaria constituyen una herramienta que puede ser bien aprovechada por los estudiantes de ingenierías en electrónica y mecatrónica. La experiencia ha demostrado que cuando toman el curso de microcontroladores ya han desarrollado aplicaciones con la tarjeta

Arduino pero desconocen la organización y funcionalidad de los dispositivos empleados. Así que, el diseño de nuevos proyectos se simplificará significativamente al reutilizar el hardware que ya han adquirido y al descargar nuevas aplicaciones en un estilo similar al que ya conocen.

6. Referencias

[1] F. Santiago Espinosa. Los Microcontroladores AVR de Atmel. 2012. Editado e Impreso por la Universidad Tecnológica de la Mixteca. ISBN: 978-607-95222-7-8.

[2] M. A. Mazidi, S. Naimi. The AVR Microcontroller and Embedded Systems using Asambly and C. 2011. Prentice Hall. ISBN 13: 978-013-800331-9.

[3] Arduino – Introduction. https://www.arduino.cc/en/Guide/Introduction. Junio de 2016.

[4] F. Reyes Cortés, J. Cid, Monjaraz. 1ª Edición. MARCOMBO, S.A. ISBN: 9788426722041

[5] ATMEL ATMega328/P, 8-Bit Microcontroller with 32Kbyte In-System programmable flash, Atmel-8271J-AVR- ATmega-Datasheet_11/2015.

[6] How To Write a Simple Bootloader For AVR In C language. http://atmega32-avr.com/how-to-write-a-simple-bootloader-for-avr-in-c-language/. Agosto de 2016.

[7] A. S. Tanenbaum, “Redes de Computadores”, 4a. ed. 2003. Ed. Prentice Hall.

[8] Paul Deitel, Harvey Deitel, Como programar en Java. 7a. ed. 2008. Ed. Pearson Educación.

[9] Giovynet Driver. www.giovynet.com/giovynetDriver_en.html. Junio de 2016.

7. Autores

M.C. Felipe Santiago Espinosa es Licenciado en Electrónica egresado de la BUAP, Maestro en Ciencias con especialidad en Electrónica por parte del INAOE y candidato a Doctor en Robótica por la UTM. Es Profesor-Investigador en la Universidad Tecnológica de la Mixteca desde 1998, perteneciendo al Instituto de

Electrónica y Mecatrónica. En el año de 2012 publicó su libro titulado “Los Microcontroladores AVR de ATMEL”.

Zenón Belarmino Martínez Cruz es estudiante de Ingeniería en Electrónica en la Universidad Tecnológica de la Mixteca, en donde está cursando el 9º semestre. En verano de 2016 realizó una estancia profesional en Soluciones Metasoftica, desempeñándose como desarrollador web.