

AIDA08SH

**AMBIENTE INTEGRADO PARA DESARROLLO Y
APRENDIZAJE CON MICROCONTROLADORES
MC9S08SH32 HABILITADOS COMO
DISPOSITIVOS CHIPBAS8SH**

Guía de inicio rápido

Por: Antonio Salvá Calleja

Profesor de carrera

Departamento de Control y Robótica

Facultad de Ingeniería

UNAM

Mayo de 2016

Índice

1. Componentes del sistema AIDA08SH	3
2. Tarjeta para desarrollo MINICON_08SH	4
3. Identificación del puerto serie en la PC para trabajar con AIDA08SH	5
4. Inicio de una sesión de trabajo con el sistema AIDA08SH	6
5. Funcionalidad de los botones de interacción de la ventana de edición	11
6. Mapa de memoria de los dispositivos CHIPBAS8SH	19
7. Acciones al reset de un dispositivo CHIPBAS8SH	19
8. Ejecución autónoma de programas en un dispositivo CHIPBAS8SH	19
9. Captura y ejecución de programas en BASIC	21
9.1 Perfilamiento de memoria para fines del compilador MINIBAS8A	22
9.2. Reporte al usuario de errores en tiempo de ejecución	31
10. Cambio de MCU destino a manejar con el PUMMA_EST	33
11. Funcionalidad de los <i>jumpers</i> y postes presentes en la tarjeta MINICON_08SH	37
12. Conectores presentes en la tarjeta MINICON_08SH	38
13. Funcionalidad de componentes relevantes que no son conectores, postes o chips	39
14. Interfazado entre la computadora anfitriona y la tarjeta MINICON_08SH empleando el módulo IUUM	39
15. Versiones del firmware de base para los dispositivos CHIPBAS8SH	41
15.1 Líneas de puerto y puntos de acceso a periféricos disponibles para el usuario en la tarjeta MINICON_08SH	42
15.2 Testificación al RESET de la versión del firmware de base presente	45
15. Referencias	46

1 Componentes del sistema AIDA08SH

Los componentes funcionales básicos del Ambiente Integrado para Desarrollo y Aprendizaje alrededor del MCU de FREESCALE MC9S08SH32, habilitado como dispositivo CHIPBAS8SH, véase [4], se muestran en la figura 1. Este sistema está integrado por cuatro componentes los cuales son:

- Tarjeta destino (Target) MINICON_08SH, basada en un MCU MC9S08SH32 habilitado como dispositivo CHIPBAS8SH.
- Módulo de Interfaz Serie Universal para tarjetas MINICON (ISUM).
- Enlace serie entre la computadora anfitriona y el módulo ISUM, esto puede ser un simple cable RS232; o bien, un adaptador USB-SERIE.
- Computadora anfitriona (Host) donde se ejecuta un software manejador

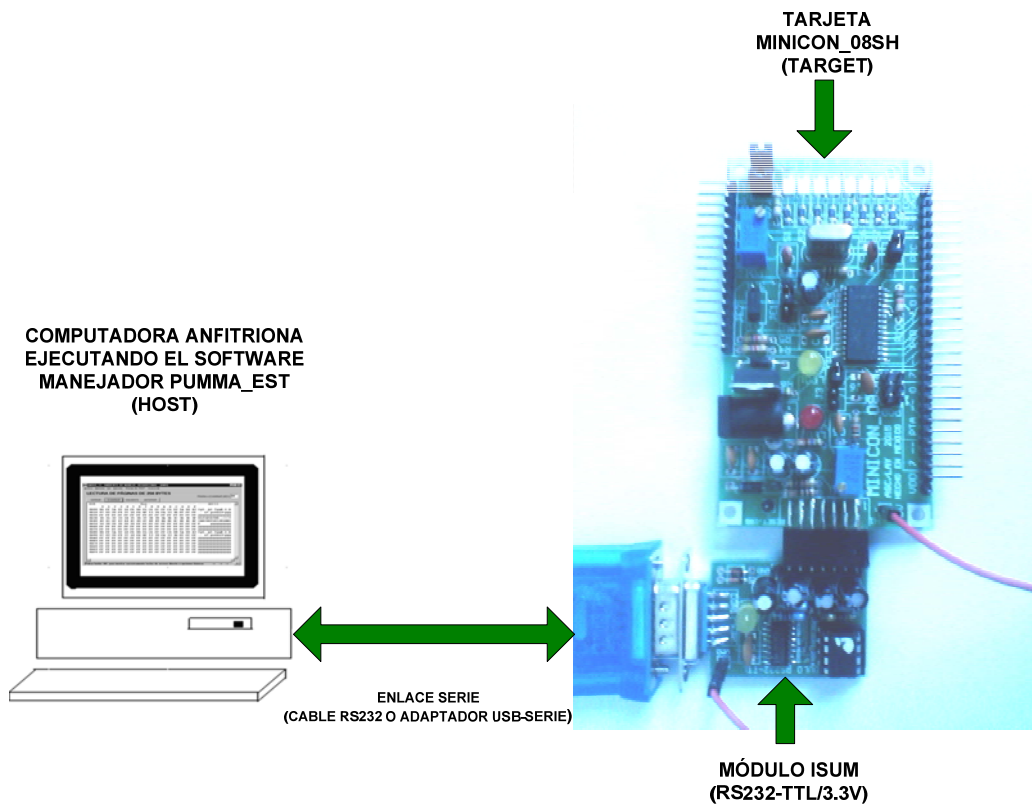


Fig 1. Sistema para desarrollo y aprendizaje AIDA08SH

El software manejador contiene, entre otras, las siguientes facilidades:

- Ensamblador cruzado denominado ENS08.
- Compilador cruzado de BASIC denominado MINIBAS8A. Mediante el cual se puede generar código ejecutable en un dispositivo CHIPBAS8XX (XX = GP32, GT, SH, JM y QG) a partir de código fuente en BASIC.
- Compilador cruzado de BASIC denominado SBAS8 que contempla el desarrollo de programas bajo el paradigma de uso de procedimientos de tipo 'Function', 'Sub' y 'Subint'. Este último para el manejo de interrupciones.

El software manejador puede configurarse para que el MCU destino sea otro, como podrían ser otros miembros de la familia hcs08 de FREESCALE; o bien, algunos miembros de la familia MSP430 de Texas Instruments. Por ejemplo, si el MCU destino fuera el MCU MC9S08GT60 de FREESCALE, el software manejador se debe configurar de modo que el chip destino sea un dispositivo CHIPBAS8GT60, y al ambiente para desarrollo integrado por los componentes mostrados en la figura 1 se le denominaría AIDA08GT.

Esta guía está focalizada a la descripción de la funcionalidad básica del sistema de desarrollo de la figura 1.

Para detalles acerca de la sintaxis del ensamblador soportado por ENS08, y lo propio acerca del lenguaje BASIC soportado por el compilador MINIBAS8A, pueden verse los capítulos 5 y 7 de la referencia [1]. Por lo que toca a la estructuración de programas compilables con el compilador SBAS8, véase [5].

Mediante el software manejador, se pueden efectuar, entre otras, las siguientes acciones sobre el MCU presente en la tarjeta:

- Ensamblar un programa en lenguaje ensamblador presente en la ventana del editor.
- Ensamblar y ejecutar en RAM un programa en lenguaje ensamblador presente en la ventana del editor.
- Ensamblar y ejecutar en memoria no volátil, un programa en lenguaje ensamblador presente en la ventana del editor.
- Compilar, bajo MINIBAS8A, un programa fuente en BASIC presente en la ventana del editor.
- Ejecutar de inmediato el programa fuente en BASIC presente en la ventana del editor; habiendo sido éste compilado bajo MINIBAS8A.
- Compilar, bajo SBAS8, un programa fuente en BASIC presente en la ventana del editor.
- Ejecutar de inmediato el programa fuente en BASIC presente en la ventana del editor; habiendo sido éste compilado bajo SBAS8.
- Borrar la memoria no volátil del MCU disponible para el usuario.
- Examinar la memoria del MCU presente en la tarjeta.
- Cargar y ejecutar un archivos S19 que contiene código de máquina propio del MCU presente en la tarjeta, el cual haya sido generado por alguna otra herramienta de compilación o ensamble, como podría ser el compilador cruzado de lenguaje C presente en el software CodeWarrior de FREESCALE.

2. Tarjeta para desarrollo MINICON_08SH

En la figura 2 de muestra la tarjeta MINICON_08SH. En la referencia [3] puede verse el diagrama esquemático de la tarjeta.

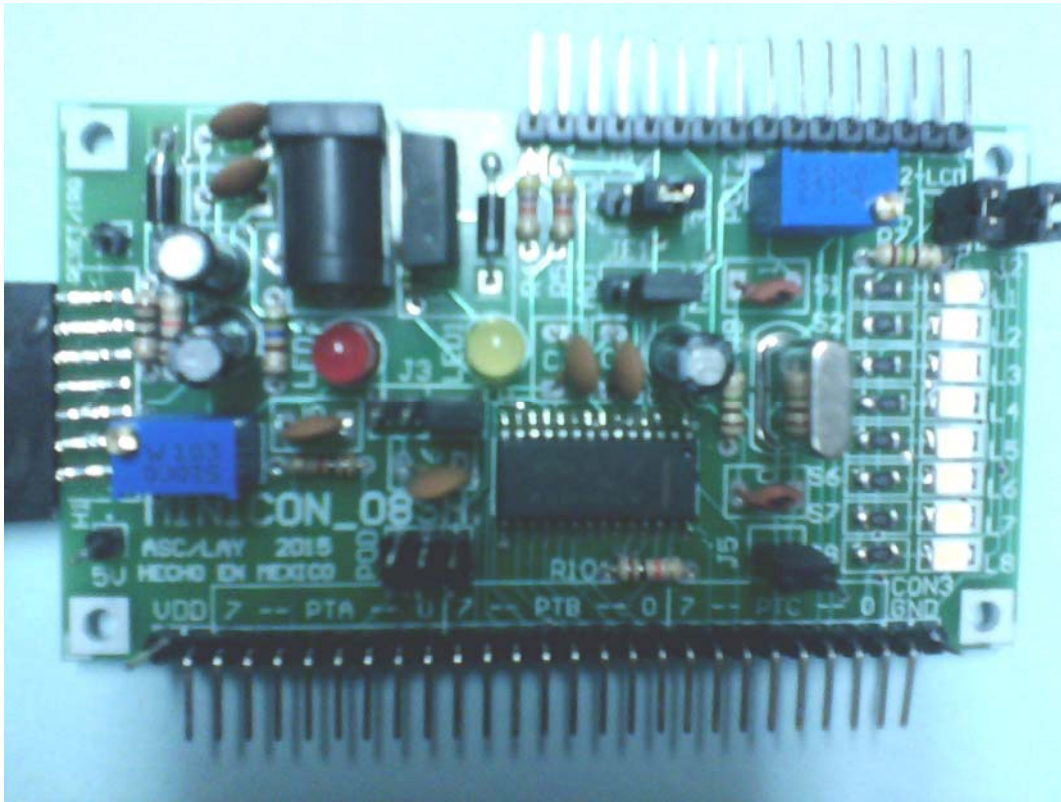


Fig 2. Tarjeta MINICON_08SH

En esta guía se detallan aspectos funcionales básicos propios del hardware de la tarjeta y de su software manejador.

3. Identificación del puerto serie en la PC para trabajar con AIDA08SH

Antes de iniciar el uso del el sistema AIDA08SH, se debe identificar un puerto serie físico o virtual disponible en la computadora donde corre el software manejador; para ello se emplea el administrador de dispositivos de WINDOWS. En la figura 3 se muestra la ventana presentada por éste, para una PC que tiene un puerto serie físico denotado como COM1, y un puerto serie virtual denotado como COM5, validado con una interfaz USB-SERIE, para la cual ya se ha instalado correctamente el *driver* asociado.

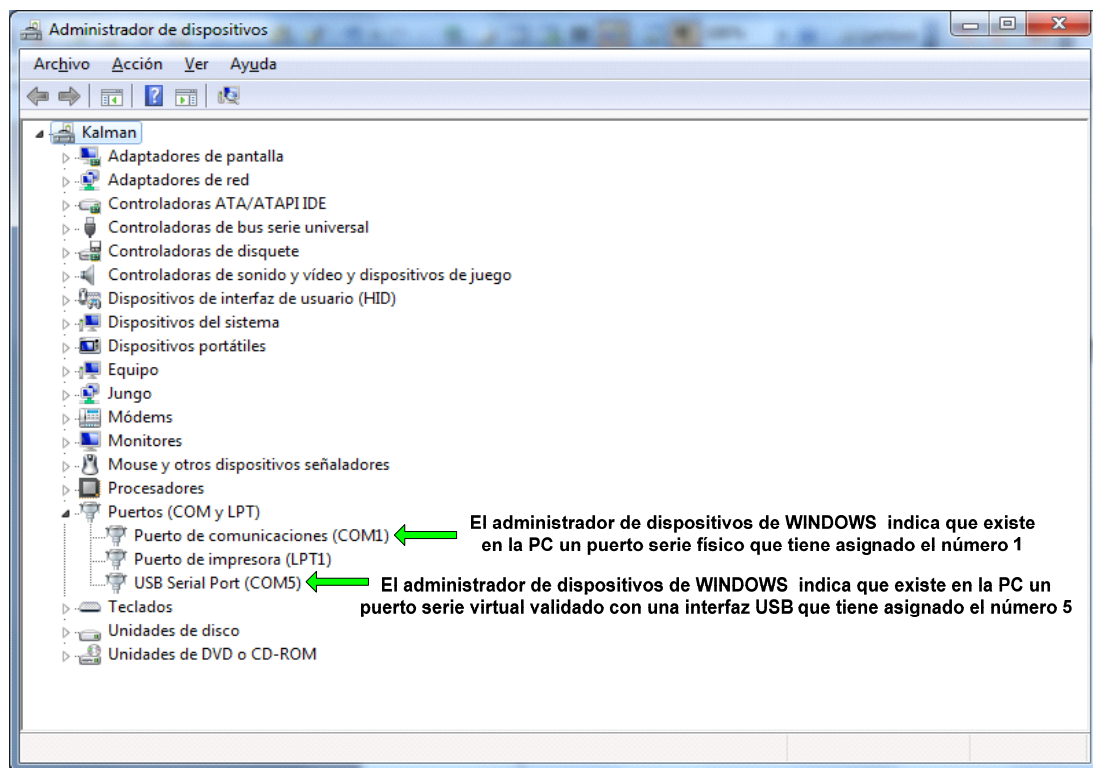


Fig 3. Ventana del administrador de dispositivos de WINDOWS para una PC que cuenta con dos puertos serie, uno fisico y el otro virtual, denotados respectivamente como COM1 y COM5.

Para fines del software manejador se podría usar cualquiera de los puertos identificados por el administrador de dispositivos de WINDOWS, siempre y cuando que el que se seleccione no esté siendo utilizado por otro programa que corra al mismo tiempo que el software manejador.

Es importante destacar que una vez que se ha identificado el puerto serie a usar, si no hay cambios en la PC, la identificación ya no será necesaria para futuras sesiones de trabajo con AIDA08SH, ya que el puerto serie a emplear por parte del software manejador se puede predeterminar como se verá más adelante en esta guía.

4. Inicio de una sesión de trabajo con el sistema AIDA08SH

Para iniciar una sesión de trabajo con el sistema para desarrollo mostrado en la figura 1 se deben efectuar los siguientes seis pasos:

PASO 1. Con la tarjeta MINICON_08SH desenergizada efectuar las siguientes acciones:

- Conectar a la tarjeta MINICON_08SH al conector de 7 hilos macho presente en el módulo ISUM.
- Conectar al conector db9 hembra presente en el módulo ISUM al conector db9 macho que contiene el cable RS232 empleado, o bien, el adaptador USB-SERIE utilizado.
- Conectar el otro extremo del adaptador USB-SERIE o del cable RS232 a la PC.

- Mediante un puente hembra – hembra interconectar los postes denominados ‘VDD’ presentes en la tarjeta MINICON_08SH y el módulo ISUM.

En la figura 4 se muestra el conexionado básico descrito aquí.

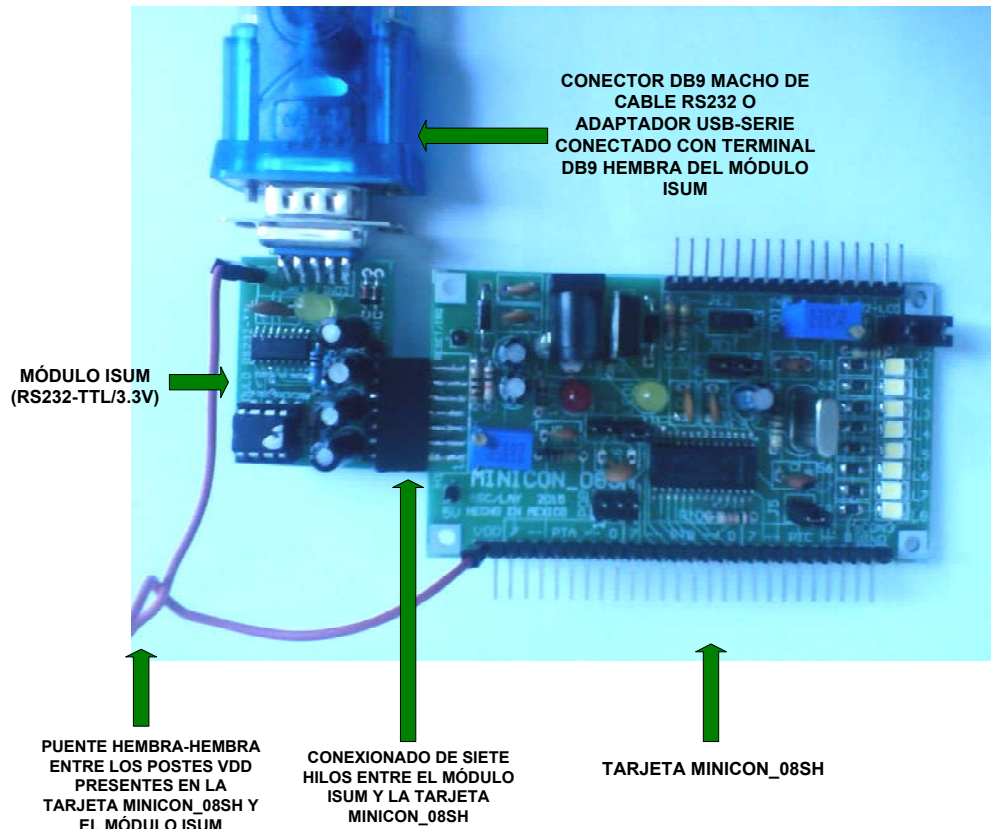


Fig 4. Conexionado básico para operar la tarjeta MINICON_08SH.

Cabe señalar que si se usa un adaptador USB-SERIE, antes de ejecutar el software manejador el *driver* del adaptador ya deberá estar correctamente instalado en la PC; además, el número de puerto serie virtual que valida el adaptador, ya deberá estar identificado por el usuario, como se detalla en la sección 3, ya que el software manejador lo solicita al inicio de la sesión. PUMMA_EST soporta que el número de puerto serie a emplear esté comprendido entre uno y dieciséis.

PASO 2. Asegurarse que el *jumper* JE1, presente en la tarjeta, esté en su posición default (MON).

PASO 3. Asegurarse que el *jumper* JE2, presente en la tarjeta, esté en su posición (7.0).

PASO 4. Energizar la tarjeta MINICON_08SH. Al hacer esto se deberá observar un parpadeo de cadencia rápida en el ledpum (LED1) presente en ésta. Testificando esto que la tarjeta puede ser manejada desde la PC mediante el software manejador PUMMA_EST o PUMMA_08+.

PASO 5. Ejecutar en la PC el software manejador PUMMA_EST. Al iniciarse éste aparecerá el dialogo mostrado en la figura 5, en donde se indica al usuario el tipo de tarjeta y chip destino preconfigurado para operar PUMMA_EST.

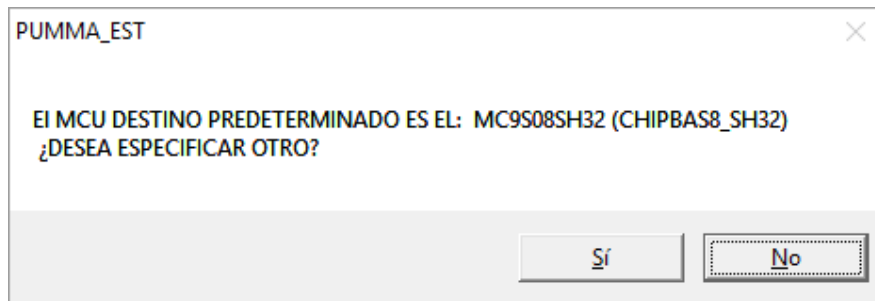


Fig 5. Dialogo que indica el tipo de MCU destino de la familia hcs08 preconfigurado para operar PUMMA_EST.

Si como se muestra en la figura 5, PUMMA_EST indica al usuario que el MCU destino es un dispositivo CHIPBAS8SH, el usuario simplemente deberá oprimir el botón <No>. Por otra parte, si el dialogo indica que el MCU predeterminado no es un dispositivo CHIPBAS8SH como se ejemplifica en la figura 6; el usuario deberá oprimir el botón <Sí>, después de esto PUMMA_EST presentará una interfaz mediante la cual el usuario puede cambiar el MCU destino predeterminado de modo que éste sea un dispositivo CHIPBAS8SH.

Una descripción de la secuencia de pasos a seguir para cambiar el MCU predeterminado se describe más adelante en la sección 9 de esta guía.

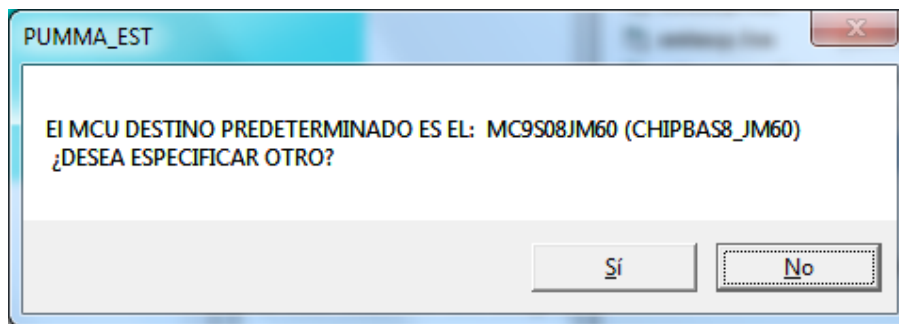


Fig 6. Ejemplo de dialogo que indica al usuario que el MCU predeterminado no es un dispositivo CHIPBAS8SH.

PASO 6. Después de que el usuario oprime el botón <No> en el dialogo de la figura 5, aparecerá un dialogo como el mostrado en la figura 7, donde PUMMA_EST indica el puerto serie en la PC preconfigurado para operar PUMMA_EST.

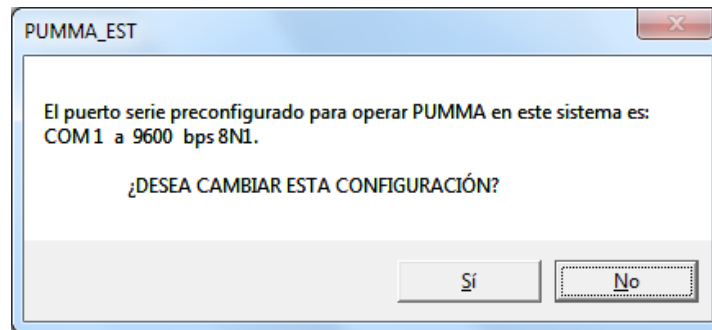


Fig 7. Dialogo mostrado por PUMMA_EST indicando el puerto serie de la PC preconfigurado para operar AIDA08SH.

Si el número de puerto serie preconfigurado coincide con el propio del puerto serie, físico o virtual, preidentificado por el usuario en la PC empleada para hacer el enlace, el usuario deberá oprimir el botón <No> en el dialogo de la figura 7. Después de esto PUMMA_EST pasa al paso 7, en otro caso se deberá oprimir el botón <Sí> lo que hará que se pase al paso 6B.

PASO 6B. PUMMA_EST pedirá al usuario el número de puerto a preconfigurar, véase la figura 8; éste deberá ser el asociado con un puerto serie funcional predientificado previamente por el usuario, véase la sección 3.

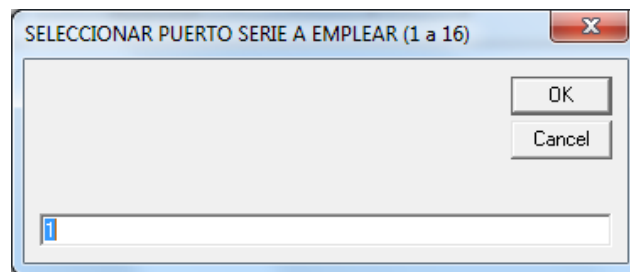


Fig 8. Requerimiento de número de puerto serie a emplear para el enlace serie

Después de lo anterior PUMMA_EST requerirá la velocidad del enlace en bits/seg (bps), véase la figura 9. Aquí el usuario siempre deberá especificar 9600 bps.

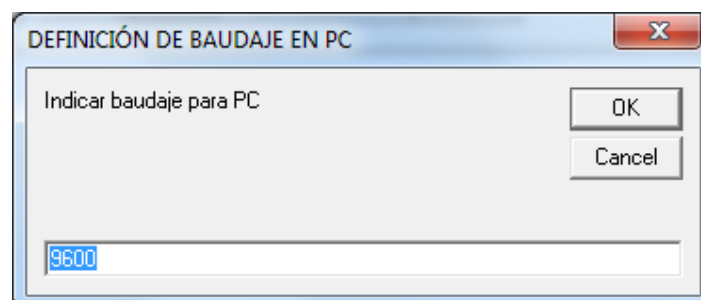


Fig 9. Requerimiento al usuario de la velocidad del enlace

Si el puerto serie especificado, cuyas características básicas se dieron en los diálogos de las figuras 8 y 9 es válido, PUMMA_EST procederá al paso 7; en otro caso,

PUMMA_EST indicará esto al usuario, véase la figura 10; después de esto, se volverá a pedir al usuario la especificación del puerto serie mediante los diálogos de las figuras 8 y 9, esto hasta por tres veces. Si el usuario agota el número de intentos para especificar un puerto serie válido, PUMMA_EST avisa esto al usuario, véase la figura 11, y pasa al paso 7. Sin embargo, debido a que no se cuenta con un puerto serie válido, el funcionamiento del software manejador será errático, y podrá abortarse la ejecución de éste cuando se ejecute un comando que implique el uso del puerto serie, como podría ser entre otros, una lectura de memoria del MCU destino. Si se da este último evento aparecerá el aviso mostrado en la figura 12, y se abortará la ejecución del software manejador después de que se oprime el botón aceptar en el dialogo de la figura 12.

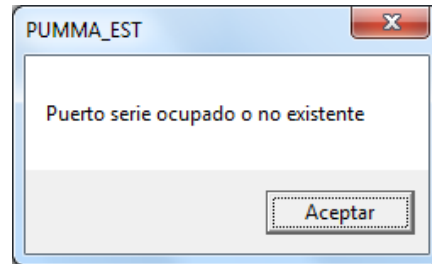


Fig 10. Indicación al usuario cuando éste ha especificado un puerto serie inválido.

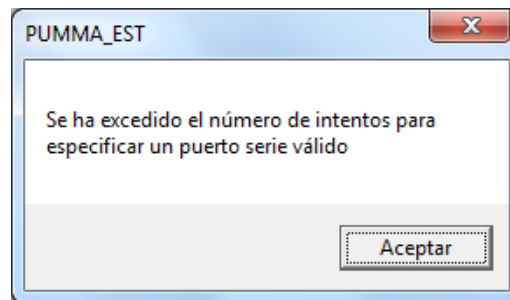


Fig 11. Indicación al usuario que se ha excedido el número de intentos para especificar el puerto serie a predeterminedar para el uso del sistema AIDA08QG.

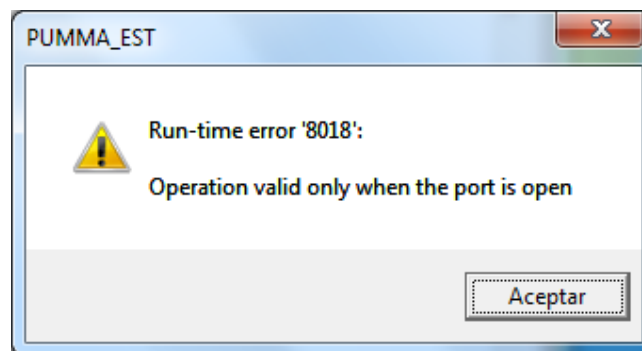


Fig 12. Aviso al usuario de un mal funcionamiento por no contar PUMMA_EST con un puerto serie válido.

PASO 7. PUMMA_EST mostrará la ventana de su editor, véase la figura 13. Ahí el usuario podrá teclear o capturar desde un archivo, un programa fuente en ensamblador, acorde con el software de ensamble ENS08; o bien, en el lenguaje BASIC validado por los compiladores MINIBAS8A o SBAS8. Una vez completado este paso estaría todo listo para iniciar la sesión de trabajo para fines de aprendizaje, desarrollo o bien simple recreación.

Cabe señalar que los archivos asociados con código en ensamblador deberán tener la extensión ‘asm’. Por otra parte, los archivos asociados con código BASIC deberán tener la extensión ‘b’. De esta forma, un archivo que contenga código en ensamblador estará denotado como np.asm y un archivo que contenga código en BASIC estará denotado como np.b, donde np es el nombre que el programador haya definido para el programa.

Para fines de la prueba y depuración de programas la ventana del editor cuenta con ocho botones de interacción denominados: ‘e-ram’, ‘e-feep’, ‘ens’, ‘e-pfbas’, ‘c-pfbas’, ‘e-pfsb8’, ‘c-pfsb8’ y ‘b-feep’. La funcionalidad de éstos se describe a continuación.



Fig 13. Ventana del editor de PUMMA_EST

5. Funcionalidad de los botones de interacción de la ventana de edición

Al oprimirse los botones de interacción presentes en la ventana del editor de PUMMA_EST, se efectúan acciones asociadas con la prueba y ejecución del programa fuente presente en el editor en un momento dado. A continuación, en la tabla 1 se describen tales acciones.

TABLA 1. ACCIONES ASOCIADAS CON LOS BOTONES DE INTERACCIÓN

BOTÓN	ACCION A EFECTUARSE AL OPRIMIRSE EL BOTÓN
<ens>	Se ensambla el programa fuente en ensamblador presente. Si no hay errores de sintaxis se generan los archivos np.lst y np.s19. Si hay errores reporta esto al usuario.
<e-ram>	Se ensambla el programa fuente en ensamblador presente. Si no hay errores de sintaxis el programa se carga en RAM y se ejecuta de inmediato en el MCU. Si hay errores se reporta esto al usuario y no se efectúa acción sobre el MCU. Es responsabilidad del programador el que el programa fuente contenga directivas que indiquen que el código de máquina ha de cargarse en memoria RAM del MCU.
<e-feep>	Se ensambla el programa fuente en ensamblador presente. Si no hay errores de sintaxis el programa se carga en memoria no volátil (FLASH) y se ejecuta de inmediato en el MCU. Si hay errores se reporta esto al usuario y no se efectúa acción sobre el MCU. Es responsabilidad del programador el que el programa fuente contenga directivas que indiquen que el código de máquina ha de cargarse en memoria FLASH del MCU.
<c-pfbas>	Se compila bajo MINIBAS8A el programa fuente en BASIC presente. Si no hay errores se generan diversos archivos de salida, entre ellos el np.s19 que contiene el código de máquina asociado con el programa fuente original. Si hay errores se reportan estos al usuario.
<e-pfbas>	Se compila bajo MINIBAS8A el programa fuente en BASIC presente. Si no hay errores el programa se carga en memoria no volátil (FLASH) y se ejecuta de inmediato en el MCU. Si hay errores se reporta esto al usuario y no se efectúa acción sobre el MCU
<c-pfsb8>	Se compila bajo SBAS8 el programa fuente en BASIC presente. Si no hay errores se generan diversos archivos de salida, entre ellos el np.s19 que contiene el código de máquina asociado con el programa fuente original. Si hay errores se reportan estos al usuario usuario.
<e-pfsb8>	Se compila bajo SBAS8 el programa fuente en BASIC presente. Si no hay errores el programa se carga en memoria no volátil (FLASH) y se ejecuta de inmediato en el MCU. Si hay errores se reporta esto al usuario y no se efectúa acción sobre el MCU
<b-feep>	Se borra el intervalo de memoria no volátil disponible para el usuario. Para un dispositivo CHIPBAS8SH este intervalo es: 8000 – D7FF (22 kb)

A continuación se muestran ejemplos de uso de AIDA08SH, para fines de la ejecución en el MCU de programas en ensamblador tanto en memoria RAM como en memoria no volátil (FLASH).

Ejemplo 1. Captura ensamble y ejecución en RAM de un programa fuente en lenguaje ensamblador

Se desea ejecutar en memoria RAM un programa en ensamblador que genere una onda cuadrada de 20 Hz en el bit ptc7 del puerto C del MCU. El programa debe originarse en la dirección de RAM \$0100. Además, se debe considerar que la frecuencia del reloj de bus para un dispositivo CHIPBAS8SH es 20 MHz ($T_b = 50 \text{ ns}$). A continuación se muestra el programa de este ejemplo:

```
ptcd equ $04
ptcdd equ $05

                                org $0100

                                bset 7,ptcdd ; ptc7 es salida.

ciclo:                          bset 7,ptcd ; ptc7 ← 1
                                bsr ret25ms
```

```

        bclr 7,ptcd ; ptc7 ← 0
        bsr ret25ms
        bra ciclo

;Subrutina de retardo, Tr = (25+10Xr)Tb
Ret25ms:    pshh
            pshx
            ldhx #$c34d
vuelta:    nop
            nop
            aix #$ff
            cphx  #$0000
            bne vuelta
            pulx
            pulh
            rts

```

En la figura 14 se muestra la ventana del editor una vez que se ha capturado el código fuente de este ejemplo y el mismo ha sido guardado en el archivo `parp_ptc7_ram.asm`. Por ser este el primer ejemplo de código en ensamblador de esta guía, para fines didácticos se detalla aquí lo que sucede al oprimirse los botones <ens> y <e-ram>.

Al oprimirse el botón <ens> en la ventana mostrada en la figura 14, si no hay errores de sintaxis aparecerá el dialogo mostrado en la figura 15 y se habrán generado los archivos de salida `parp_ptc7_ram.lst` y `parp_ptc7_ram.s19`. En los renglones del archivo ‘lst’ aparecerán, para cada una de las instrucciones del programa, a la derecha el código fuente original, y a la izquierda el código de máquina con la dirección de carga de éste. En lo que toca al archivo ‘s19’, éste contiene el código de máquina asociado con el programa fuente presentado en un formato estándar establecido por FREESCALE. En las figuras 16 y 17 se muestran, en la ventana del editor, los archivos ‘lst’ y ‘s19’ generados por el ensamblador.

```

J:\hcs08\parp_ptc7_ram.asm
Archivo  Editar  Buscar  Opciones  Acerca de
nvo  abr  gdr  ctr  cpr  pgr  e-ram  e-leep  ens  e-plbas  e-plbas  e-plsb8  c-plsb8  b-leep  min

ptd equ $04
ptd equ $05

        org $0100

        bset 7,ptd ; ptc7 es salida.

ciclo:
        bset 7,ptd ; ptc7 <-- 1
        bsr ret25ms
        bclr 7,ptd ; ptc7 <-- 0
        bsr ret25ms
        bra ciclo

;Subrutina de retardo, Tr = (25+10Xr)Tb
ret25ms:
        pshh
        pshx
        ldhx #c34d
vuelta:
        nop
        nop
        aix #fff
        cphx #0000
        bne vuelta
        pulx
        pulh
        rts

```

Figura 14. Ventana del editor con el archivo ‘asm’ asociado al programa del ejemplo 1

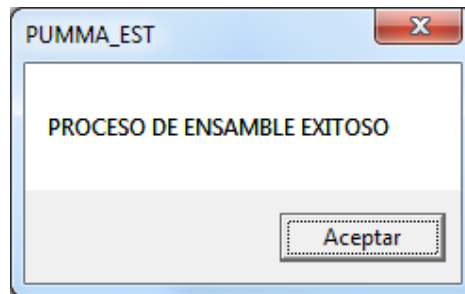


Figura 15. Dialogo presentado al usuario después de que se ha oprimido el botón <ens> cuando el programa presente en ensamblador no tiene errores de sintaxis.

En caso de que existan errores de sintaxis en el código fuente en ensamblador presente en la ventana del editor, al oprimirse el botón <ens> el proceso de ensamble se detendrá al encontrarse el primer error en el programa; en la figura 18 se indica como PUMMA_EST muestra al usuario la ocurrencia del primer error encontrado en el programa presente. Para fines ilustrativos se colocaron mnemónicos inválidos en los renglones 15 y 16 del programa. Nótese que el ensamblador muestra la ocurrencia sólo del primer error encontrado. Para regresar al modo de operación normal del editor bastará con hacer click sobre cualquier punto del área de edición en la ventana mostrada en la figura 18. Si el usuario deseara saber el número asociado con un determinado renglón del código fuente, bastará que haga doble clic sobre éste, enseguida a esto aparecerán en la parte superior derecha del marco del área de edición los números de renglón y carácter implicados. Para fines ilustrativos, en la figura 19 aparece como se indica en la ventana del editor el número asociado con un renglón sobre el cual se ha dado un doble click, cuando éste es el que contiene el error indicado en la figura 18.

```

J:\hcs08\parp_ptc7_ram.LST
Archivo  Editar  Buscar  Opciones  Acerca de
nvo  abr  gdr  ctr  cpr  pgr  e-ram  e-feep  ens  e-plbas  e-plbas  e-plsb8  c-plsb8  b-feep  min

J:\hcs08\parp_ptc7_ram.asm  Ensamblado con ENS08 el 28/03/2016 1:45:22

0000          ptdc equ $04
0000          ptdcd equ $05
0000
0100                      org $0100
0100
0100 1E05                      bset 7,ptcdd ; ptc7 es salida.

0102 1E04          ciclo:      bset 7,ptcd ; ptc7 <-- 1
0104 AD06                      bsr ret25ms
0106 1F04                      bclr 7,ptcd ; ptc7 <-- 0
0108 AD02                      bsr ret25ms
010A 20F6                      bra ciclo

010C          ;Subrutina de retardo, Tr = (25+10Xr)Tb
010C 8B          ret25ms:      pshh
010D 89                      pshx
010E 45C34D        ldhx #c34d
0111 9D          vuelta:      nop
0112 9D                      nop
0113 AFFF          aix #fff
0115 650000        cphx #0000
0118 26F7          bne vuelta
011A 88          pulx
011B 8A          pulh
011C 81          rts

011D

```

Figura 16. Ventana del editor con el archivo ‘lst’ asociado al programa del ejemplo 1

```

J:\hcs08\parp_ptc7_ram.S19
Archivo  Editar  Buscar  Opciones  Acerca de
nvo  abr  gdr  ctr  cpr  pgr  e-ram  e-feep  ens  e-plbas  e-plbas  e-plsb8  c-plsb8  b-feep  min

S11301001E051E04AD061F04AD0220F68B8945C3EF
S11001104D9D9DAFFF65000026F7888A8194
S9030000FC

```

Figura 17. Ventana del editor con el archivo ‘s19’ asociado al programa del ejemplo 1

```

J:\hcs08\parp_ptc7_ram.asm
Archivo  Editar  Buscar  Opciones  Acerca de
nvo  abr  gdr  ctr  cpr  pgr  e-ram  e-feep  ens  e-pfbas  e-pfbas  e-pfsb8  c-pfsb8  b-feep  min
ptd equ $04
ptdd equ $05
org $0100
bset 7,ptdd ; ptc7 es salida.
ciclo:
bset 7,ptd ; ptc7 <-- 1
bsr ret25ms
bclr 7,ptd ; ptc7 <-- 0
bsr ret25ms
bra ciclo
;Subrutina de retardo, Tr = (25+10Xr)Tb
ret25ms:
pshq
psht
ldhx #c34d
vuelta:
nop
nop
aix #fff
cphx #0000
bne vuelta
puls
puls
rts

```

Figura 18. Detallado al usuario acerca del primer error encontrado por el ensamblador en un programa fuente.

```

J:\hcs08\parp_ptc7_ram.asm
Archivo  Editar  Buscar  Opciones  Acerca de
nvo  abr  gdr  ctr  cpr  pgr  e-ram  e-feep  ens  e-pfbas  e-pfbas  e-pfsb8  c-pfsb8  b-feep  min
LP=15 CN=344
ptd equ $04
ptdd equ $05
org $0100
bset 7,ptdd ; ptc7 es salida.
ciclo:
bset 7,ptd ; ptc7 <-- 1
bsr ret25ms
bclr 7,ptd ; ptc7 <-- 0
bsr ret25ms
bra ciclo
;Subrutina de retardo, Tr = (25+10Xr)Tb
ret25ms:
pshq
psht
ldhx #c34d
vuelta:
nop
nop
aix #fff
cphx #0000
bne vuelta
puls
puls
rts

```

Figura 19. Ejemplo de indicación del número de renglón sobre el que se ha hecho doble click.

Para fines de la ejecución del programa de este ejemplo, bastará con oprimir el botón <e-ram> en la ventana mostrada en la figura 14, si no hay errores de sintaxis

PUMMA_EST carga y ejecuta de inmediato el programa implicado en este ejemplo, apreciándose un parpadeo visible en el led conectado al bit ptc7 del MCU. Además, se verá que el LEDPUM (LED1), que testifica la disposición del MCU para recibir comandos desde el software manejador estará apagado, esto se debe a que el programa de este ejemplo tomo el control del MCU; bajo esta circunstancia no se podrán efectuar acciones sobre el MCU desde el software manejador. Para concluir la ejecución del programa y regresar al MCU al modo de recepción de comandos bastará con dar un reset en la tarjeta, para ello bastará con conectar y desconectar los dos postes asociados con el *Jumper* J3, después de esto se volverá a observar un parpadeo de cadencia rápida en el LEDPUM.

Ejemplo 2. Ejecución en memoria no volátil del programa del ejemplo 1

Se desea ejecutar desde la memoria no volátil del MCU el programa del ejemplo 1; para ello bastará asociar con la directiva 'org' al principio de éste una dirección donde exista, de ahí en adelante, disponibilidad de memoria no volátil para el usuario. Sabiendo que, para un dispositivo CHIPBAS8SH el intervalo de memoria no volátil disponible para el programador es [\$8000-\$d7ff], véase más adelante la tabla 2, la dirección inicial del programa podría ser el inicio del intervalo citado. En la figura 20 se muestra la ventana del editor de PUMMA_EST con el código fuente en ensamblador de este ejemplo, el cual ha sido guardado en el archivo parp_ptc7_flash.asm.

```

J:\hcs08\parp_ptc7_flash.asm
Archivo  Editar  Buscar  Opciones  Acerca de
nvo  abr  gdr  ctr  cpr  pgr  e-ram  e-feep  ens  e-ptbas  c-ptbas  e-pfsb8  c-pfsb8  b-feep  min

ptcd equ $04
ptcdd equ $05

        org $8000

        bset 7,ptcdd ; ptc7 es salida.

ciclo:   bset 7,ptcd ; ptc7 <-- 1
        bsr ret25ms
        bclr 7,ptcd ; ptc7 <-- 0
        bsr ret25ms
        bra ciclo

;Subrutina de retardo, Tr = (25+10Xr)Td
ret25ms: pshh
        pshx
        ldhx #$c34d
vuelta:  nop
        nop
        aix #$ff
        cphx #$0000
        bne vuelta
        pulx
        pulh
        rts

```

Figura 20. Código fuente del ejemplo 2 una vez que éste ha sido capturado

Para ejecutar el programa de este ejemplo, bastará con oprimir el botón <e-flash> presente en la ventana del editor mostrada en la figura 20. Después de esto, si no hay errores de sintaxis, PUMMA_EST cargará en memoria no volátil del MCU destino el código de máquina y éste se ejecutará de inmediato. En caso de que haya errores de sintaxis, se reportará esto al usuario como se describe en el ejemplo 1.

Para aclarar ideas, a continuación se muestra el contenido del archivo `parp_ptb7_flash.lst` generado por el ensamblador. Nótese que el intervalo de localidades de memoria no volátil que ocupará el programa es [\$8000-\$801c], lo cual quiere decir que el programa cabe dentro del intervalo de localidades de memoria no volátil disponibles para el programador en un dispositivo CHIPBAS8SH.

Contenido del archivo `parp_ptb7_flash.lst` generado por el ensamblador

J:\hcs08\parp_ptb7_flash.asm Ensamblado con ENS08 el
28/03/2016 08:01:45 p.m.

```

0000          ptcdd equ $04
0000          ptcdd equ $05
0000
8000                                org $8000
8000
8000 1E05                                bset 7,ptcdd ; ptb7 es salida.

8002 1E04          ciclo:          bset 7,ptcd ; ptb7 <-- 1
8004 AD06                                bsr ret25ms
8006 1F04                                bclr 7,ptcd ; ptb7 <-- 0
8008 AD02                                bsr ret25ms
800A 20F6                                bra ciclo
800C          ;Subrutina de retardo, Tr = (25+10Xr)Tb
800C 8B          ret25ms:          pshh
800D 89                                pshx
800E 459C3D          ldhx #$c34d
8011 9D          vuelta:          nop
8012 9D                                nop
8013 AFFF          aix #$ff
8015 650000          cphx #$0000
8018 26F7          bne vuelta
801A 88          pulx
801B 8A          pulh
801C 81          rts

801D

```

TABLA DE SÍMBOLOS

ETIQUETAS

CICLO	8002
RET50MS	800C
VUELTA	8011

ASIGNACIONES EQU

PTCD	\$04
PTCDD	\$05

6. Mapa de memoria de los dispositivos CHIPBAS8SH

En la tabla 2 se detalla el mapa de memoria de un dispositivo CHIPBAS8SH, esto acorde con los espacios ocupados por el firmware de base y lo propio acerca del mapa de memoria propio del MCU, véase el capítulo 4 de [2].

TABLA 2. MAPA DE MEMORIA DE LOS DISPOSITIVOS CHIPBAS8QG

INTERVALO (HEX)	CONTENIDO
0000-007F	Zona 1 de registros de control y operación (Direct page registers).
0800-047F	Memoria RAM (1024 localidades).
0480-17FF	Zona no implementada (4992 localidades).
1800-185F	Zona 2 de registros de control y operación (High page registers).
1860-7FFF	Zona no implementada (26528 localidades).
8000-D7FD	Memoria de usuario. (22526 localidades). Aquí es donde el programador puede colocar el programa que este diseñando en un momento dado.
D7C2-D7FD	Direcciones de colocación de vectores de interrupción de usuario.
D7FE-D7FF	Vector de reset de la aplicación (VRA).
8000- FFFF	Zona protegida contra escritura y borrado. Contiene el firmware de base que habilita al MCU como dispositivo CHIPBAS8SH.

Nota: Si la aplicación no usa interrupciones el intervalo de memoria de usuario es: 8000-D7FD. Si la aplicación usa interrupciones, el intervalo de memoria de usuario es: 8000-D7XX, donde D7XX es la dirección inmediata anterior, a la de colocación del vector de interrupción de usuario, asociado con la interrupción de menor prioridad que se use en la aplicación.

7. Acciones al reset de un dispositivo CHIPBAS8SH

Al darse un reset en un dispositivo CHIPBAS8SH, acorde con el estado del *jumper* excluyente JE1 se efectúan las acciones descritas en la tabla 3.

TABLA 3. ACCIONES AL RESET PARA UN DISPOSITIVO CHIPBAS8QG

POSICIÓN DE JE1	ACCIÓN AL RESET
POSICION DEFAULT (MON)	Se pasa al receptor de comandos que habilita al MCU para ser manejado desde la computadora anfitriona, donde corre el software manejador. Esto se testifica con un parpadeo de cadencia rápida en el ledpum (LED1).
POSICIÓN ALTERNA (AUT)	Se pasa a ejecutar en forma autónoma un programa previamente grabado, con el formato adecuado para esto, en la memoria no volátil

Nota: Para el reset bajo la posición default (MON), el firmware de base (FB) inicializa el MCU de modo que el COP está deshabilitado, el pin PTA5/IRQ/TCLK/RESET funciona únicamente como reset, el pin PTA4/ACMPO es funcional como línea de puerto; además, y el modo STOP está habilitado. Para efectuar esto el FB escribe un byte \$21 en el registro SOPT1, véase la página 69 de [2]. Para el reset bajo la posición (AUT) se deja al programador la escritura inicial al registro SOPT1, de esta forma, la aplicación podrá usar o no el COP, además de tenerse la capacidad de inicializar el MCU acorde con las necesidades de la aplicación.

8. Ejecución autónoma de programas en un dispositivo CHIPBAS8SH

Para ejecutar de manera autónoma un programa, éste deberá tener los siguientes tres bloques funcionales:

- **Bloque 1.** Aquí se efectúa la escritura inicial del registro SOPT1 con un byte que hace que se deshabilite el COP (watch dog), **esto en caso de que la aplicación no haga uso de esta facilidad del MCU.** El código de este bloque deberá ser lo primero que se ejecute del programa. Para deshabilitar el COP habría que escribir en el registro SOPT1, cuya dirección es \$1802, un byte para el cual los bits 7 y 6 sean cero. Además, si se desea que el pin

PTA5/IRQ/TCLK/RESET funcione únicamente como reset el bit menos significativo del byte a escribir en el registro SOPT1 deberá ser uno; por otra parte, si se desea que el pin **PTA5/IRQ/TCLK/RESET** funcione como: PTA5 ó IRQ ó TCLK, dicho bit deberá ser cero, véase la página 69 de [2]. Cabe señalar aquí que SOPT1 es uno de los registros de configuración básica del MCU, y que éste solo puede ser escrito una vez después de un reset. En caso de que la aplicación si use el COP el byte a escribir en el registro SOPT1 deberá tener los bits 7 y 6 diferentes de 00. El valor del byte a escribir en SOPT1 se deberá definir acorde con la configuración básica del MCU gobernada por el registro SOPT1; además, para configurar como va a operar el COP deberá escribirse lo requerido en el registro SOPT2, véase lo propio acerca del COP y del registro SOPT2 en el capítulo 5 de [2].

- **Bloque 2.** Código propio del programa a ejecutar de manera autónoma.
- **Bloque 3.** Al final del programa se deberá colocar código, que haga que los dos bytes que representan a la dirección inicial de éste, sean colocados respectivamente en las direcciones propias para ello. De acuerdo con la tabla 2, para un dispositivo CHIPBAS8SH estas direcciones son: \$d7fe y \$d7ff.

De acuerdo con lo explicado acerca del bloque uno de un programa a ejecutarse de forma autónoma, si éste no usa el COP, y además se requiera que el pin PTA5/IRQ/TCLK/RESET funcione únicamente como reset, el código del bloque uno podría ser:

```
lda #$21    ;Deshabilita COP y pin PTA5/IRQ/TCLK/RESET,
sta $1802   ;funciona únicamente como reset.
```

Por otra parte, si la aplicación a ejecutarse de manera autónoma no usa el COP, y además se requiere que el pin PTA5/IRQ/TCLK/RESET funcione como: PTA5 ó IRQ ó TCLK, el código del bloque uno podría ser:

```
lda #$20    ;Deshabilita COP y pin PTA5/IRQ/TCLK/RESET,
sta $1802   ;funciona como: PTA5 ó IRQ ó TCLK
```

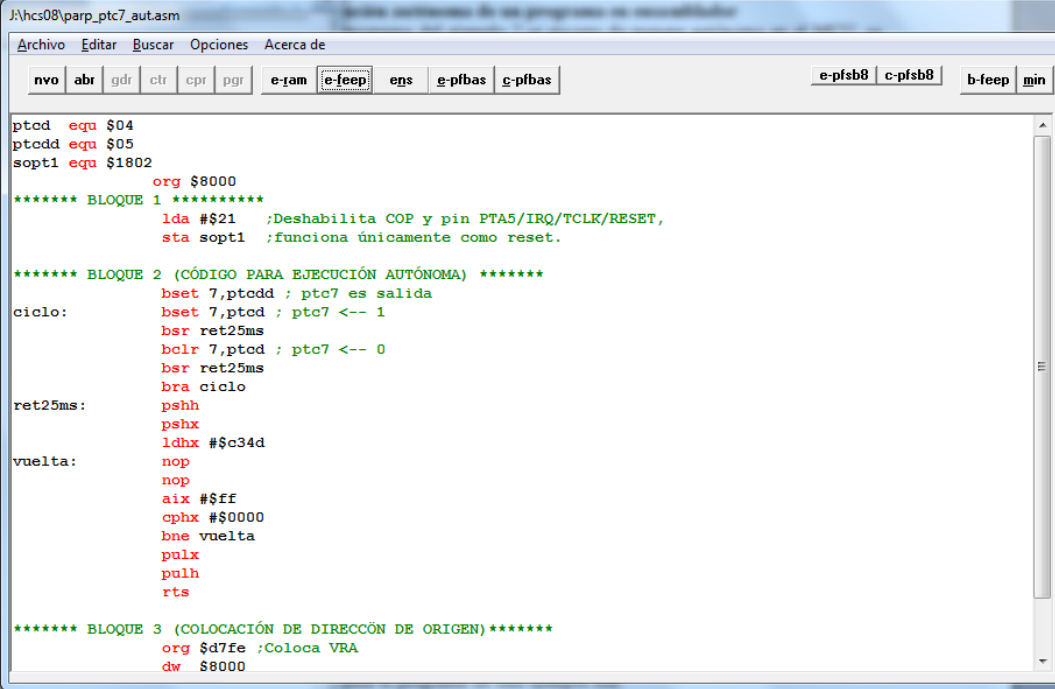
De esta forma, acorde con lo definido en la tabla 3, los pasos a seguir para ejecutar de manera autónoma un programa son:

- 1.- Con el *jumper* JE1 en su posición default (MON), grabar en la memoria no volátil del MCU el programa con el formato aquí descrito; empleando para ello facilidades de PUMMA_EST.
- 2.- Desenergizar la tarjeta MINICON_08SH.
- 3.- Pasar el *jumper* JE1 a su posición alterna (AUT).

Al energizar de nuevo la tarjeta se ejecutará de manera autónoma el programa implicado.

Ejemplo 3. Ejecución autónoma de un programa en ensamblador

Se desea que el programa del ejemplo 2 se ejecute de manera autónoma en el MCU, se supone que esta aplicación no usa el COP. De esta forma, al código propio del programa se deberá agregar los bloques de código 1 y 3 descritos al inicio de esta la sección. En la figura 21 se muestra la ventana del editor con el programa del ejemplo 2, al cual se ha agregado código de modo que éste tenga el formato para que pueda ser ejecutado de forma autónoma en el MCU. El programa ha sido guardado en el archivo `parp_ptc7_aut.asm`.



```
J:\hcs08\parp_ptc7_aut.asm
Archivo  Editar  Buscar  Opciones  Acerca de
nvo  abr  gdr  ctr  cpr  pgr  e-tam  e-feep  ens  e-pfbas  c-pfbas  e-pfsb8  c-pfsb8  b-feep  min

ptcd equ $04
ptcdd equ $05
sopt1 equ $1802
      org $8000
***** BLOQUE 1 *****
      lda #$21 ;Deshabilita COP y pin PTA5/IRQ/TCLK/RESET,
      sta sopt1 ;funciona únicamente como reset.

***** BLOQUE 2 (CÓDIGO PARA EJECUCIÓN AUTÓNOMA) *****
cielo: bset 7,ptcdd ; ptc7 es salida
      bset 7,ptcd ; ptc7 <-- 1
      bsr ret25ms
      bclr 7,ptcd ; ptc7 <-- 0
      bsr ret25ms
      bra cielo
ret25ms: pshh
      pshx
      ldhx #$c34d
vuelta: nop
      nop
      aix #$ff
      cphx #$0000
      bne vuelta
      pulx
      pulh
      rts

***** BLOQUE 3 (COLOCACIÓN DE DIRECCIÓN DE ORIGEN)*****
      org $d7fe ;Coloca VRA
      dw $8000
```

Figura 21. Programa `parp_ptc7_aut.asm` para ejecución autónoma.

De esta forma, acorde con lo explicado en la sección 8, los pasos a seguir para ejecutar de manera autónoma el programa de este ejemplo son:

- 1.- Con el *jumper* JE1 en su posición default (MON), y con el programa capturado tal como se muestra en la figura 21, oprimir el botón <e-feep>, lo que hará que éste se grave en la memoria no volátil del MCU a partir de la dirección \$8000. Enseguida a lo cual pasara a ejecutarse de inmediato.
- 2.- Desenergizar la tarjeta MINICON_08SH.
- 3.- Pasar el *jumper* JE1 a su posición alterna (AUT).

Al energizar de nuevo la tarjeta se ejecutará de manera autónoma el programa de este ejemplo.

9. Captura y ejecución de programas en BASIC

En esta sección se explican diversos aspectos básicos relacionados con el uso del compilador MINIBAS8A presente en el software manejador PUMMA_EST. Para

apoyo ilustrativo se usarán tres ejemplos, para una mejor comprensión de éstos puede verse el capítulo 7 de la referencia [1]. Los puntos a tratar son:

- Perfilamiento de memoria para fines del compilador MINIBAS8A.
- Pasos a seguir para compilar un programa en BASIC. Esto ilustrado con el ejemplo 4.
- Pasos a seguir para ejecutar en el MCU destino un programa en BASIC. Esto ilustrado con el ejemplo 5.
- Reporte al usuario de errores en tiempo de ejecución.
- Pasos a seguir para la ejecución autónoma de un programa en BASIC. Esto ilustrado con el ejemplo 6.

9.1 Perfilamiento de memoria para fines del compilador MINIBAS8A

Para fines de la generación de código ejecutable en el MCU empleando el compilador MINIBAS8A, se requiere delimitar los intervalos de memoria volátil y no volátil que emplee el programa en código de máquina asociado con un determinado programa fuente en BASIC. Para fines de un dispositivo CHIPBAS8SH estos intervalos deberán ser congruentes con lo indicado en la tabla 2. Por ejemplo, el inicio del intervalo de colocación del código ejecutable podría ser el origen del área de memoria no volátil disponible para el programador (\$8000); o bien, una dirección superior que no rebase la dirección (\$d7ff). Por otra parte, el origen del intervalo de memoria no volátil para fines de la colocación de las variables que use el programa podría ser la dirección inicial del área de RAM (\$0080); o bien, una dirección superior que no rebase el intervalo de memoria no volátil del MCU.

Para fines de efectuar la delimitación de intervalos de memoria mencionados en el párrafo anterior, PUMMA_EST cuenta con un submenú denominado “Memoria en chip destino”, que es parte del menú opciones presente en la ventana del editor, véase la figura 22.

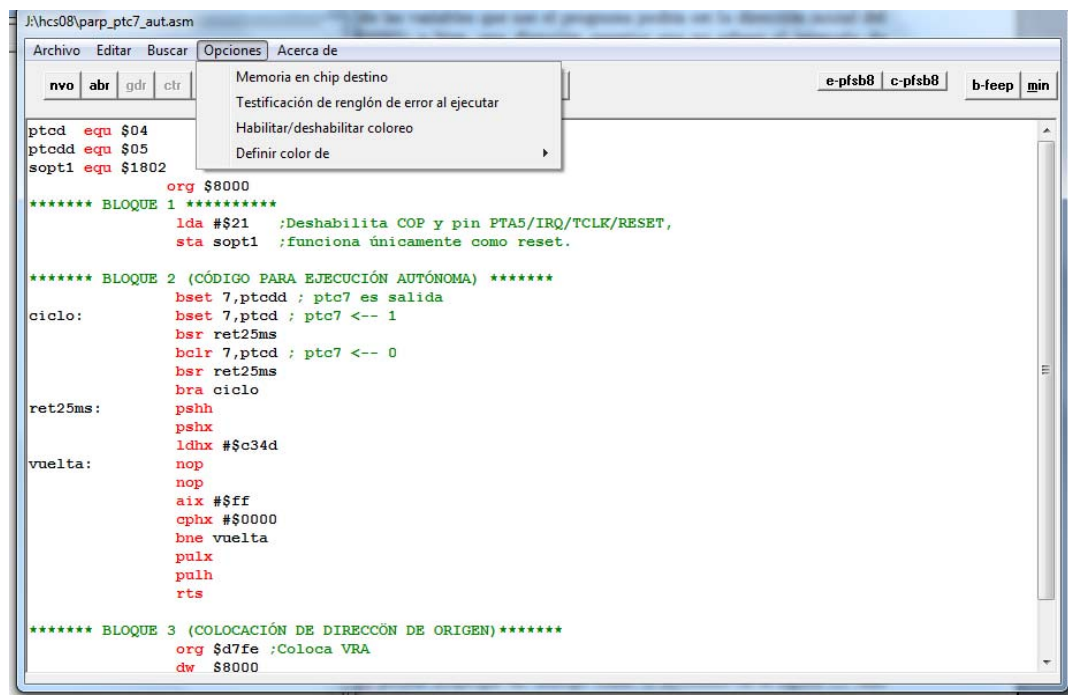


Fig 22. Submenús presentes en el menú opciones de la ventana del editor.

Cuando el usuario invoca el submenú “Memoria en chip destino” mostrado en la figura 22, PUMMA_EST podría desplegar un dialogo como el mostrado en la figura 23. Ahí se aprecia que el perfilamiento de memoria no corresponde con el requerido para un dispositivo CHIPBAS8SH, ya que la dirección inicial de colocación de código es \$182c y el origen del intervalo para colocación de variables, denotado como “Dirección inicial de datos” es \$0087. Este perfilamiento de memoria preconfigurado previamente es el que correspondería a un dispositivo CHIPBAS8GT, que tiene intervalos de memoria volátil y no volátil diferentes a los propios de un dispositivo CHIPBAS8SH.

Fig 23. Posible dialogo desplegado al invocarse el submenú “Memoria en chip destino”.

Para cambiar y preconfigurar el perfil de memoria se deberán llevar a cabo los siguientes pasos:

1. Abrir el combo presente en el dialogo de la figura 23. Al hacer esto aparecerán los distintos chips destino para los cuales se puede preconfigurarse el perfil de memoria. Véase la figura 24.
2. Seleccionar en la lista mostrada en la figura 24 el chip destino. Véase la figura 25, nótese que en ésta los dispositivos CHIPBAS8SH están denotados con el alias CHIPBAS8_SH32/16.
3. Oprimir el botón aplicar. Después de esto se deberá observar que el perfilamiento de memoria ya es el adecuado para un dispositivo CHIPBAS8SH. Véase la figura 26.
4. Oprimir el botón aceptar lo que hará que el perfil de memoria ya sea el propio de un dispositivo CHIPBAS8SH.

GENERADOR DE CÓDGO EN ENSAMBLADOR HC08

CONFIGURACIÓN BÁSICA DE MEMORIA PARA EL MCU DESTINO

SP INICIAL

DIRECCIÓN INICIAL DE CÓDIGO (DIC)

DIRECCIÓN INICIAL DE DATOS (DID)

DIRECCIÓN FINAL DE DATOS (DFD)

SELECCIONAR TIPO DE MCU DESTINO

CHIPBAS8_GP32
 CHIPBAS8_GB/T60
 CHIPBAS8_MM128/64
 CHIPBAS8_QG8
 CHIPBAS8_JM60
 CHIPBAS8_SH32/16
 CHIPBAS8_SH8

☐ SP INICIAL Y DIC MODIFICABLES
☐ DFD ACORDE CON INTERRUPCIONES EN BASIC

APLICAR ACEPTAR

Fig 24. Dialogo donde PUMMA_EST muestra los diversos chips destino posibles

GENERADOR DE CÓDGO EN ENSAMBLADOR HC08

CONFIGURACIÓN BÁSICA DE MEMORIA PARA EL MCU DESTINO

SP INICIAL

DIRECCIÓN INICIAL DE CÓDIGO (DIC)

DIRECCIÓN INICIAL DE DATOS (DID)

DIRECCIÓN FINAL DE DATOS (DFD)

SELECCIONAR TIPO DE MCU DESTINO

☐ SP INICIAL Y DIC MODIFICABLES
☐ DFD ACORDE CON INTERRUPCIONES EN BASIC

APLICAR ACEPTAR

Fig 25. Dialogo donde ya se ha seleccionado perfilar la memoria de un dispositivo CHIPBAS8SH.

GENERADOR DE CÓDIGO EN ENSAMBLADOR HC08

— CONFIGURACIÓN BÁSICA DE MEMORIA PARA EL MCU DESTINO —

SP INICIAL:

SELECCIONAR TIPO DE MCU DESTINO:

DIRECCIÓN INICIAL DE CÓDIGO (DIC):

DIRECCIÓN INICIAL DE DATOS (DID):

DIRECCIÓN FINAL DE DATOS (DFD):

☐ SP INICIAL Y DIC MODIFICABLES

☐ DFD ACORDE CON INTERRUPCIONES EN BASIC

Fig 26. Dialogo mostrado al oprimir el botón aplicar en el dialogo de la figura 25.

Cabe señalar que si al invocar el submenú “Memoria en chip destino”, aparece un dialogo como el mostrado en la figura 27, donde los datos en las cajas de texto de la izquierda sean los mostrados en la figura 26, esto indicaría que el perfilamiento de memoria preconfigurado corresponde al propio de un dispositivo CHIPBAS8SH. En este caso, simplemente se deberá oprimir el botón <aceptar>. O bien, si hay duda de que ese perfilamiento sea el propio de un dispositivo CHIPBAS8SH, habría que seguir los cuatro pasos descritos anteriormente para cambiar el perfilamiento de memoria.

Una vez que se ha preconfigurado el perfil de memoria, éste prevalecerá hasta que el mismo sea cambiado por el usuario debido a que se esté trabajando con otro dispositivo CHIPBAS8.

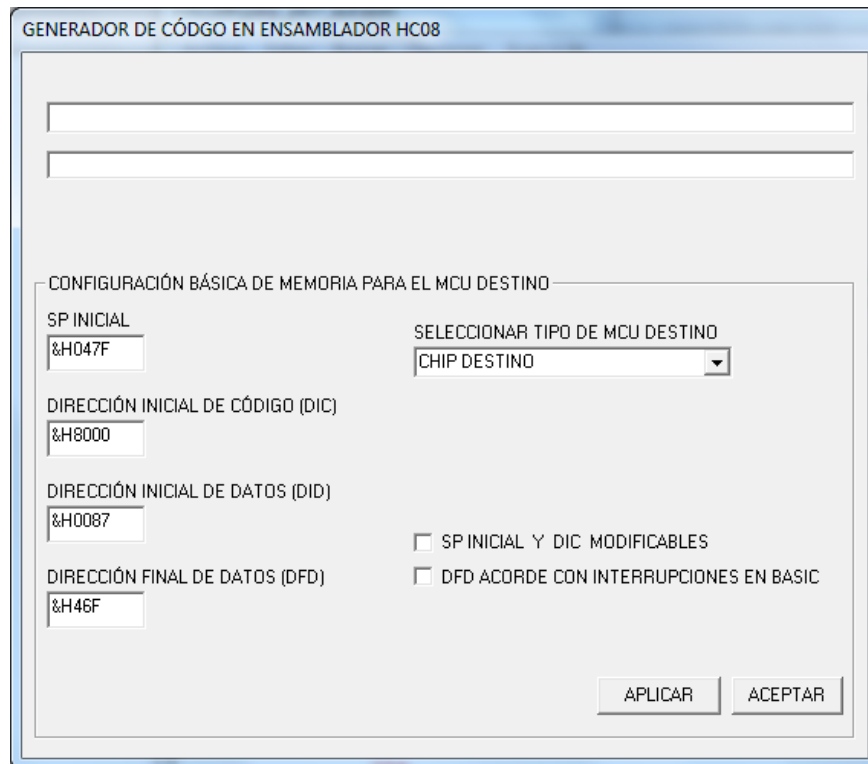


Fig 27. Dialogo presentado por PUMMA_EST al invocarse el submenú “Memoria en chip destino”, cuando el perfilamiento de memoria preconfigurado es el propio de un dispositivo CHPBAS8SH.

Ejemplo 4. Compilación de un programa en BASIC

En este ejemplo se ilustra como se compila un programa en BASIC, empleando las facilidades propias para este fin presentes en el software PUMMA_EST. Para ello se presenta un programa que al ejecutarse pide al usuario los tres coeficientes de una ecuación cuadrática, se obtienen las dos raíces desplegándose éstas en la consola de interfaz; ésta para fines del código generado por MINIBAS8A, debe ser un emulador de terminal serie. PUMMA_EST cuenta con un emulador de terminal serie que se abre al pulsar el submenú “Invocar emulador de terminal”, presente en el menú “Archivo” de la ventana del editor. El archivo que contiene el programa se denominó solec2g_sh32.b. Es importante recalcar que los archivos fuente de BASIC compilables con MINIBAS8A deben tener la extensión “b”. Una vez que se han hecho las operaciones y los resultados han sido desplegados en la consola, el programa vuelve a pedir otro trío de coeficientes y repite el proceso.

En la figura 28 se muestra el programa fuente de este ejemplo presente en la ventana del editor de PUMMA_EST. Para compilarlo se deberá oprimir el botón <c-pfbas>.

Si no hay errores léxicos, de sintaxis o semánticos en el programa fuente, después de un tiempo que dependerá del tamaño del programa, PUMMA_EST mostrará el dialogo mostrado en la figura 29. Si hay errores se reportarán estos al usuario. Para fines ilustrativos se colocaron dos errores en el programa fuente, uno en la línea 28 donde se cambió la palabra reservada ‘endif’ por la cadena ‘endig’, y el otro en la línea 30 donde se quitó la sentencia ‘wend’ que cierra el estructura ‘while’, véase la figura 30. En la figura 31 se muestra como son reportados estos errores al usuario.

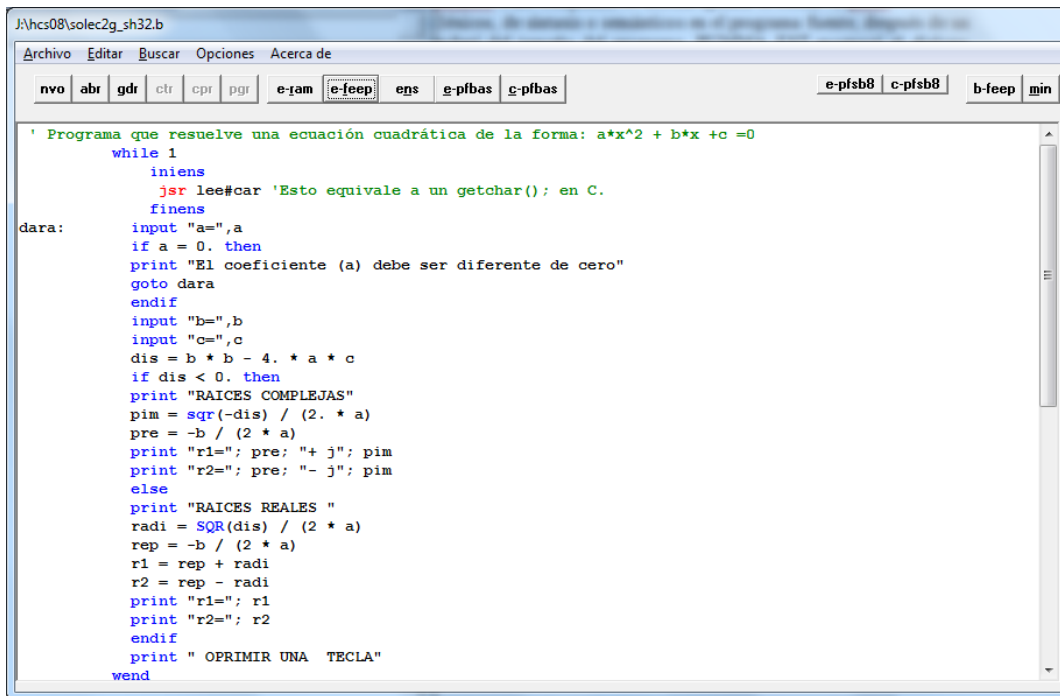


Fig 28. Programa del ejemplo 4 presente en la ventana del editor de PUMMA_EST

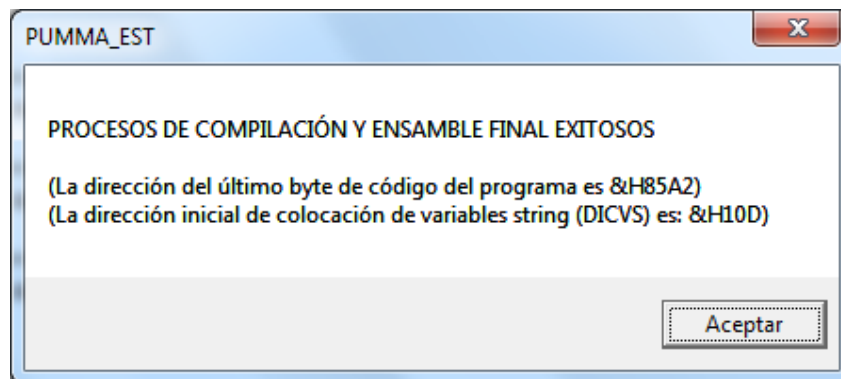


Fig 29. Reporte al usuario acerca del éxito de la compilación efectuada al oprimirse el boton <c-pfbas>.

```

J:\vcs08\solec2g_sh32.b
Archivo  Editar  Buscar  Opciones  Acerca de
nvo  abr  gdr  ctr  cpr  pgr  e-iam  e-feep  ens  e-plbas  e-plbas  e-plsb8  c-plsb8  b-feep  min

' Programa que resuelve una ecuación cuadrática de la forma: a*x^2 + b*x + c = 0
while 1
  iniens
  jsr lee#car 'Esto equivale a un getchar(); en C.
  finens
dara:  input "a=",a
      if a = 0. then
      print "El coeficiente (a) debe ser diferente de cero"
      goto dara
      endif
      input "b=",b
      input "c=",c
      dis = b * b - 4. * a * c
      if dis < 0. then
      print "RAICES COMPLEJAS"
      pim = sqr(-dis) / (2. * a)
      pre = -b / (2 * a)
      print "r1="; pre; "+ j"; pim
      print "r2="; pre; "- j"; pim
      else
      print "RAICES REALES "
      radi = SQR(dis) / (2 * a)
      rep = -b / (2 * a)
      r1 = rep + radi
      r2 = rep - radi
      print "r1="; r1
      print "r2="; r2
      endig
      print " OPRIMIR UNA  TECLA"
    wend

```

Fig. 30 Programa fuente del ejemplo 4 con errores en los renglones 28 y 30

Se encontraron 3 errores a continuación indicados:

CERRAR

```

+ ESTE ES EL ENCABEZADO DEL ARCHIVO DE ERRORES DE SINTAXIS Y SEMÁNTICOS
Declaración 'while' sin 'wend' en el renglón 2

-----> while 1
-----
Encabezado de 'if' tipo bloque sin 'endif' en el renglón 14

-----> if dis < 0. then
-----
Expresión sin asignación a variable simple o arreglo en el renglón 28

endig
-----

```

Fig 31 Reporte al usuario de los errores del programa mostrado en la figura 30

Para detalles acerca de la gramática y diversos aspectos funcionales del lenguaje BASIC soportado por el compilador cruzado MINIBAS8A, véase el capítulo 7 de [1].

Ejemplo 5. Ejecución de un programa en BASIC

En este ejemplo se ilustra como ejecutar en un dispositivo CHIPBAS8SH un programa en BASIC. Esto bajo el sistema para desarrollo AIDA08SH, véase la figura 1. Para esto se empleará el programa del ejemplo 4. Los pasos a seguir son:

1. Con la tarjeta MINICON_08SH desenergizada, asegurarse que ésta esté adecuadamente ligada a la computadora PC anfitriona vía enlace serie, y que el *jumper* excluyente JE1 esté en la posición (MON).
2. Energizar la tarjeta MINICON_08SH. Se deberá observar que el LEDPUM (LED1) parpadea, testificando esto que el dispositivo CHIPBAS8SH está en condiciones de recibir comandos desde la computadora anfitriona donde corre el software manejador PUMMA_EST.
3. Bajo la ventana del editor de PUMMA_EST abrir el archivo que contiene el programa fuente en BASIC que se desea ejecutar en el MCU. Para este ejemplo es el archivo solec2g_sh32.b, véase la figura 28.
4. Oprimir el botón <e-pfbas>. Después de esto transcurrirá un intervalo de tiempo que dependerá del tamaño del programa que se esté compilando en un momento dado. Si no hay errores, PUMMA_EST procederá a cargar en la memoria no volátil del MCU el código de máquina asociado. Una vez que ha terminado la carga del programa, PUMMA_EST hará que éste se ejecute de inmediato. Se deberá observar que el LEDPUM se apaga. Si se encontraron errores, estos se reportan al usuario como se ha ilustrado en el ejemplo 4, y no se efectúa la acción de carga sobre el MCU.
5. Si el programa implicado contiene sentencias ‘input’ y/o ‘print’, como es el caso de este ejemplo, para validar la consola de interfaz se deberá abrir el emulador de terminal presente en PUMMA_EST. En la pantalla del emulador se deberá testificar lo propio acerca de las diversas acciones de interacción entre el usuario y el programa que se ejecute en el MCU.

En la figura 32 se muestra la ventana del emulador de Terminal una vez que este ha sido invocado.

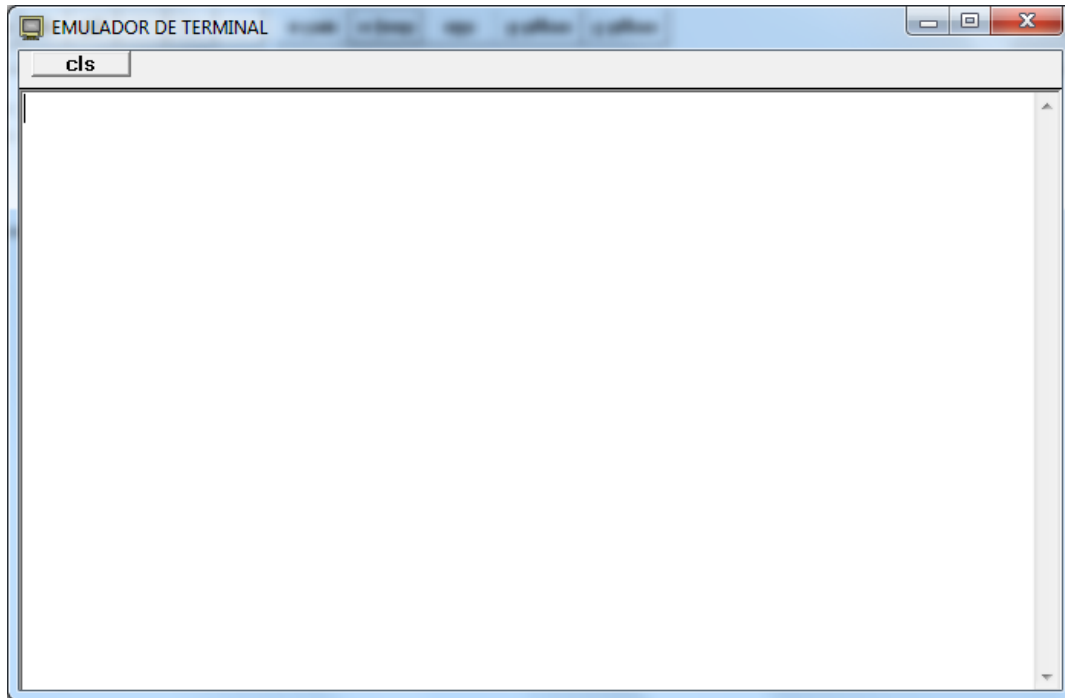
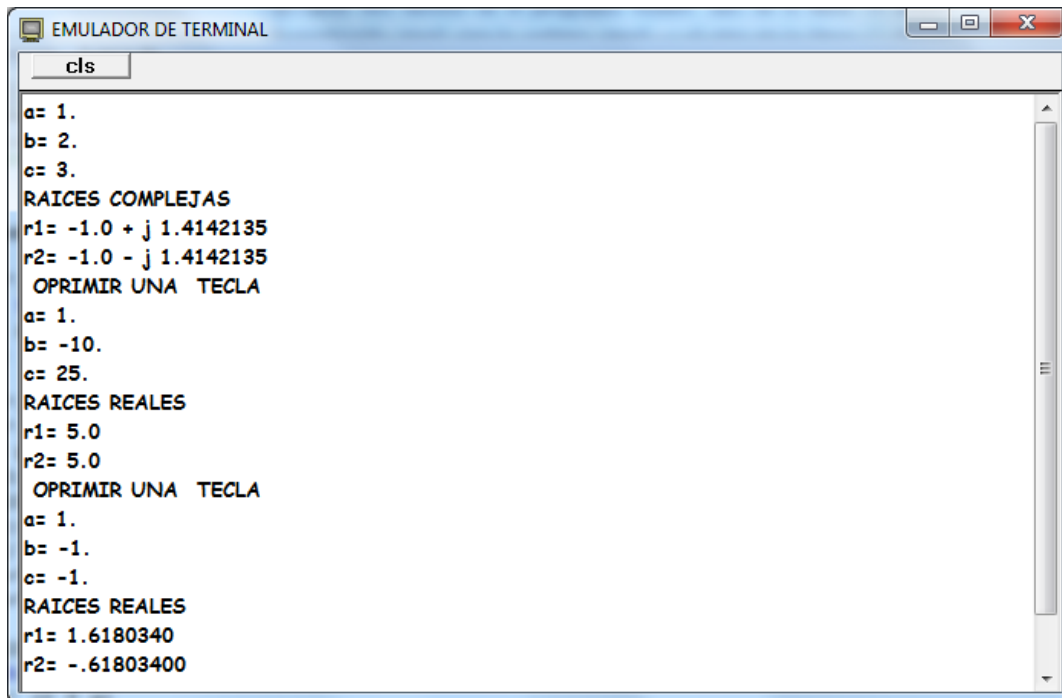


Fig 32. Ventana del emulador de terminal de PUMMA_EST

Para fines de este ejemplo, una vez que el programa se esté ejecutando en el MCU y se haya abierto el emulador de terminal, el usuario deberá oprimir cualquier tecla de ésta; después de esto el programa pedirá los valores de las variables reales de precisión sencilla a, b y c, efectuará las operaciones aritméticas y decisiones requeridas para obtener las dos raíces de la ecuación cuadrática, cuyos coeficientes son los que se han especificado, mostrando en la consola el valor de las raíces obtenidas. Después, vuelve a pedir otro juego de coeficientes y se repite el proceso. En la figura 33 se muestra la ventana del emulador de Terminal cuando se está ejecutando el programa de este ejemplo.



```
EMULADOR DE TERMINAL
cls
a= 1.
b= 2.
c= 3.
RAICES COMPLEJAS
r1= -1.0 + j 1.4142135
r2= -1.0 - j 1.4142135
OPRIMIR UNA TECLA
a= 1.
b= -10.
c= 25.
RAICES REALES
r1= 5.0
r2= 5.0
OPRIMIR UNA TECLA
a= 1.
b= -1.
c= -1.
RAICES REALES
r1= 1.6180340
r2= -.61803400
```

Fig. 33. Aspecto de la interfaz de usuario al ejecutarse el programa de la figura 28.

9.2. Reporte al usuario de errores en tiempo de ejecución

Al ejecutarse en el MCU un programa pueden darse diversos errores, entre otros, que haya una división entre cero, que se de un sobreflujo en el valor de una variable numérica, etc. Cuando esto sucede, MINIBAS8A reporta en la consola el error y el número de renglón donde se originó éste, después de esto por lo regular se detiene la ejecución del programa implicado. **El único caso en que no se detiene la ejecución del programa, es cuando el error se deba a que el resultado de una suma de strings, sea un string de tamaño mayor que 32.**

Cabe señalar que, para que se indique el número de renglón del error, previamente a la carga y ejecución del programa se deberá haber checado el submenú “testificación de renglón de error al ejecutar” del menú opciones, véase la figura 22. Si el submenú aquí mencionado no está checado antes de la carga y ejecución de un programa, al darse errores en tiempo de ejecución se reportan éstos; sin embargo, el número de renglón implicado aparecerá siempre como cero. Por otra parte, si se está en la fase de prueba de un programa que no requiera el uso del emulador de terminal, pero que al ejecutarse se puedan dar errores como los aquí mencionados, es muy conveniente, para fines de depuración, tener el emulador de terminal abierto mientras el programa se ejecuta; así al darse un error en tiempo de ejecución, el usuario podrá identificar porqué, si fue el caso, se pudo haber detenido la ejecución del programa.

Para fines ilustrativos se usará una versión modificada del programa de los dos ejemplos anteriores, de modo que se pueda dar una división entre cero en tiempo de ejecución. Para ello simplemente se quita la estructura ‘if’ donde se checa si el valor del coeficiente ‘a’ es cero en cuyo caso simplemente se vuelve a pedir éste. El programa modificado se muestra en la figura 34 y se almacena en el archivo solec2g_sh32err.b

```

J:\hcs08\solec2g_sh32err.b
Archivo  Editar  Buscar  Opciones  Acerca de
nvo  abr  gdr  ctr  cpr  pgr  e-iam  e-feep  ens  e-plbas  e-plbas  e-plsb8  c-plsb8  b-feep  min

' Programa que resuelve una ecuación cuadrática de la forma: a*x^2 + b*x + c = 0
while 1
    iniens
        jsr lee#car 'Esto equivale a un getchar(); en C.
    finens
dara:    input "a=",a

        input "b=",b
        input "c=",c
        dis = b * b - 4. * a * c

        if dis < 0. then
            print "RAICES COMPLEJAS"
            pim = sqr(-dis) / (2. * a)
            pre = -b / (2 * a)
            print "r1="; pre; "+ j"; pim
            print "r2="; pre; "- j"; pim
        else
            print "RAICES REALES "
            radi = SQR(dis) / (2 * a)
            rep = -b / (2 * a)
            r1 = rep + radi
            r2 = rep - radi
            print "r1="; r1
            print "r2="; r2
        endif

        print " OPRIMIR UNA  TECLA"
    wend

```

Fig. 34. Programa de los ejemplos 4 y 5 con la estructura 'if' inicial eliminada.

En la figura 35 se muestra el reporte presentado al usuario en la consola de interfaz, cuando se ha generado un error en tiempo de ejecución por división entre cero.

```

EMULADOR DE TERMINAL
cls

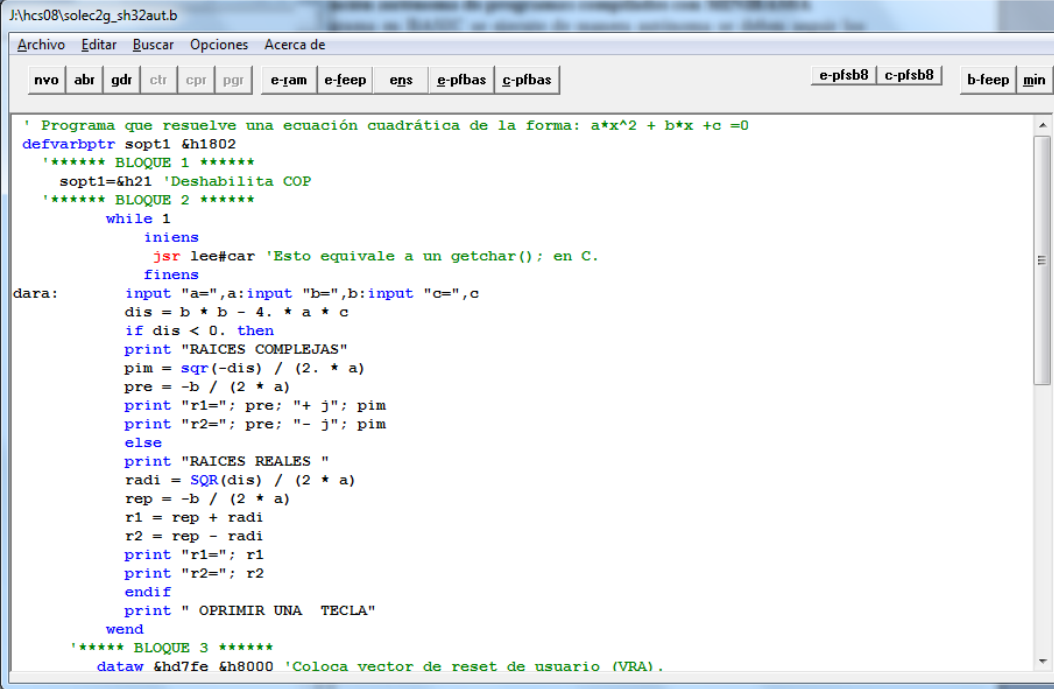
a= 1.
b= 2.
c= 3.
RAICES COMPLEJAS
r1= -1.0 + j 1.4142135
r2= -1.0 - j 1.4142135
OPRIMIR UNA  TECLA
a= 0.
b= 1.
c= 3.4
RAICES REALES
División entre cero en el renglón:
00020

```

Fig. 35. Ejemplo de reporte al usuario cuando se ha dado una división entre cero al ejecutarse programa de la figura 34.

Ejemplo 6. Ejecución autónoma de programas compilados con MINIBAS8A

Para que un programa en BASIC se ejecute de manera autónoma se deben seguir los mismos pasos para este fin descritos en la sección 8, solo que el código implicado en este caso estará en lenguaje BASIC y podrá contener incrustaciones de código en ensamblador. Para fines ilustrativos en este ejemplo se usará el programa del ejemplo 4, al que se le agrega para fines de la ejecución autónoma, los bloques uno y tres descritos en la sección 8. El programa implicado se guardó en el archivo solec2g_sh32aut.b; en la figura 35 se muestra el programa en la ventana del editor.



```
J:\hcs08\solec2g_sh32aut.b
Archivo  Editar  Buscar  Opciones  Acerca de
nvo  abr  gdr  ctr  cpr  pgr  e-ram  e-feep  ens  e-pfbas  c-pfbas  e-pfsb8  c-pfsb8  b-feep  min

' Programa que resuelve una ecuación cuadrática de la forma: a*x^2 + b*x + c = 0
defvarbptr sopt1 $h1802
'***** BLOQUE 1 *****
sopt1=$h21 'Deshabilita COP
'***** BLOQUE 2 *****
while 1
  iniens
  jsr lee#car 'Esto equivale a un getchar(); en C.
  finens
dara:  input "a=",a:input "b=",b:input "c=",c
       dis = b * b - 4. * a * c
       if dis < 0. then
       print "RAICES COMPLEJAS"
       pim = sqr(-dis) / (2. * a)
       pre = -b / (2 * a)
       print "r1="; pre; "+ j"; pim
       print "r2="; pre; "- j"; pim
       else
       print "RAICES REALES "
       radi = SQR(dis) / (2 * a)
       rep = -b / (2 * a)
       r1 = rep + radi
       r2 = rep - radi
       print "r1="; r1
       print "r2="; r2
       endif
       print " OPRIMIR UNA  TECLA"
wend
'***** BLOQUE 3 *****
dataw $hd7fe $h8000 'Coloca vector de reset de usuario (VRA).
```

Fig. 35. Programa del ejemplo 4 con los agregados para fines de su ejecución autónoma.

De esta forma, acorde con lo explicado en la sección 8, los pasos a seguir para ejecutar de manera autónoma el programa de este ejemplo son:

- 1.- Con el *jumper* JE1 en su posición default (MON), y con el programa capturado tal como se muestra en la figura 35, oprimir el botón <e-pfbas>, lo que hara que éste se grabe en la memoria no volátil del MCU a partir de la dirección \$8000. Enseguida a lo cual pasara a ejecutarse de inmediato.
- 2.- Desenergizar la tarjeta MINICON_08SH.
- 3.- Pasar el *jumper* JE1 a su posición alterna (AUT).

Al energizar de nuevo la tarjeta se ejecutará de manera autónoma el programa de este ejemplo.

10. Cambio de MCU destino a manejar con el PUMMA_EST

Si se desea cambiar el MCU destino predeterminado, al presentar PUMMA_EST al inicio de su ejecución un dialogo como los mostrados en las figuras 4 ó 5, se deberá

oprimir el botón <Sí>, después de esto aparecerá el dialogo mostrado en la figura 36. En éste se deberá indicar el software de base supuesto para el chip destino y sobre que familia de los microcontroladores hcs08 se va a trabajar. Para un dispositivo CHIPBAS8SH se deberá indicar que el MCU destino debe contener el firmware NBCP8 + BIB_MINIBAS8A que lo habilita como dispositivo CHIPBAS8, y que la familia implicada es la hcs08, véase la figura 36. Para fines ilustrativos aquí se detalla como predeterminar que el chip destino sea un dispositivo CHIPBAS8SH.

Fig 36. Dialogo para especificar software de base y familia de chip destino.

Una vez que en el dialogo de la figura 36 se haya especificado el software de base y la familia del MCU destino, se deberá oprimir el botón <SIGUIENTE>, después de esto PUMMA_EST habilita el combo titulado como 'GRUPO'. En éste el usuario deberá seleccionar el grupo 'S'; en la figura 37 se muestra el dialogo una vez que el usuario a seleccionado este grupo. Después de esto se deberá oprimir el botón <CONFIRMAR GRUPO>, enseguida a lo cual en el combo titulado como 'CPU ESPECÍFICO' se deberá definir el MCU específico. En la figura 38 se muestra el dialogo cuando el usuario ya ha definido que el MCU destino es un dispositivo CHIPBAS8SH, ahí denotado con el alias CHIPBAS8_SH32. Después de esto se deberá oprimir el botón <FINALIZAR>, lo que hará que PUMMA_EST muestre el dialogo de la figura 39 y se se cierre después de que se oprima en éste el botón <ACEPTAR>, habiéndose ya

predeterminado que el MCU destino sea un dispositivo CHIPBAS8SH, lo que tendrá efecto en la siguiente ejecución de PUMMA_EST.

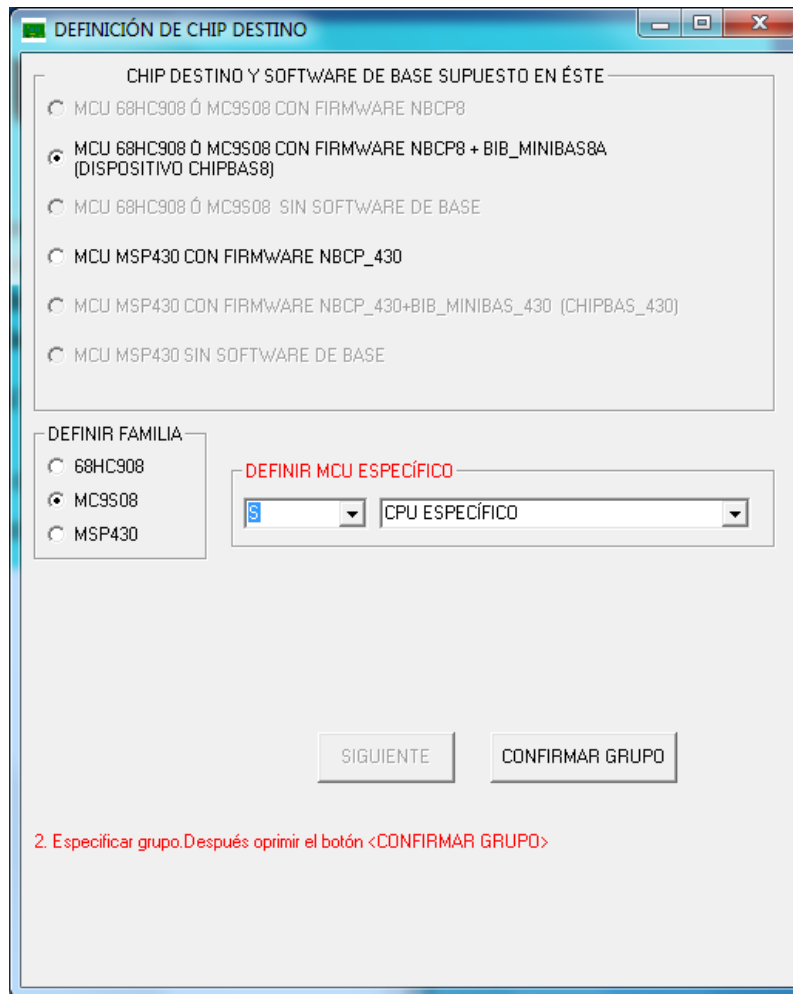


Fig. 37. Dialogo en el que ya se ha seleccionado el grupo S para el MCU destino.

En esta sección se han detallado los pasos a seguir para predeterminar que el MCU destino sea un dispositivo CHIPBAS8SH. Para predeterminar otro MCU destino sobre el cual se vaya a trabajar, la secuencia a seguir es similar con los cambios obvios en lo definido por los combos y otras alternativas presentadas al usuario en los diálogos de las figuras 36, 37 y 38.

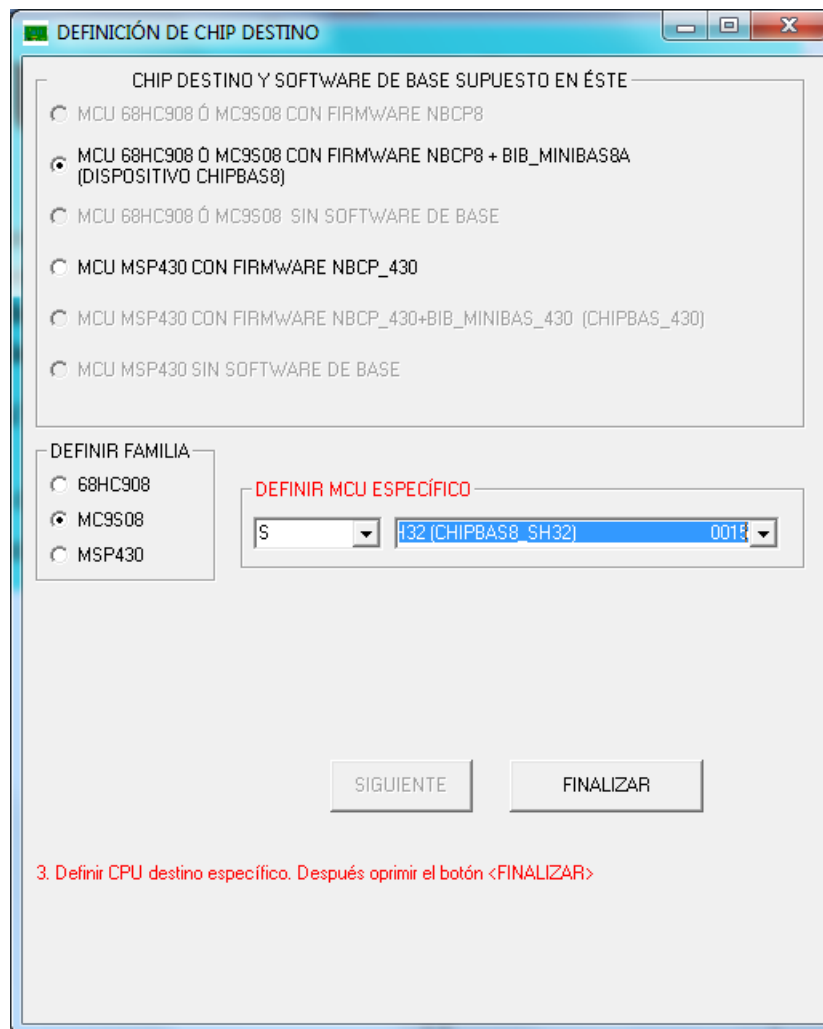


Fig. 38. Dialogo donde ya se ha definido que el MCU destino es un dispositivo CHIPBAS8SH.

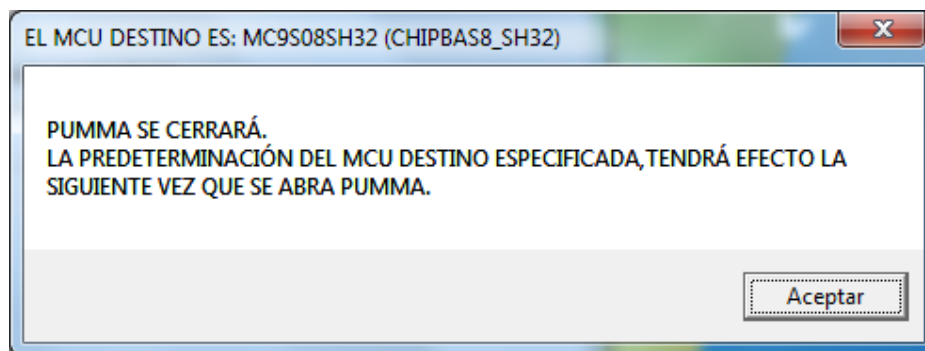


Fig. 39. Dialogo mostrado al usuario después de oprimir el botón <FINALIZAR> en el dialogo de la figura 38.

11. Funcionalidad de los *jumbers* y postes presentes en la tarjeta MINICON_08SH

En las tablas 4 y 5 se detallan respectivamente las funcionalidades propias de los *jumbers* simples y excluyentes. En la tabla 6 se detalla la funcionalidad de los postes presentes en la tarjeta. Para una mejor comprensión se sugiere ver la referencia [3].

TABLA 4. JUMPERS SIMPLES DE LA TARJETA MINICON_08SH

JUMPER	POSICIÓN DEFAULT	FUNCIONALIDAD
J1	NO COLOCADO	PIN PTB2/PIB2/SPSCK/ADP6 (18) DEL MCU ESTÁ CONECTADO A EL CURSOR DEL POTENCIOMETRO POT1
J2	COLOCADO	CONEXIÓN A TIERRA DE LOS OCHO LEDS TESTIGO DE BITS PTC0 A PTC7
J3	NO COLOCADO	CONEXIÓN A TIERRA de PIN PTA5/IRQ/TCLK/RESET (3) DEL MCU
J4	COLOCADO	LUZ DE FONFO DEL LCD
J5	COLOCADO	CONEXIÓN DE CRISTAL EXTERNO A PIN PTB7/SCL/EXTAL (9) DEL MCU

TABLA 5. JUMPERS EXCLUYENTES DE LA TARJETA MINICON_08QG

JE	FUNCIONALIDAD PARA POSICIÓN DEFAULT	FUNCIONALIDAD PARA POSICIÓN ALTERNA
JE1	Al reset el MCU ejecuta el receptor de comandos presente en el software de base. (Posición indicada como MON)	Al reset se pasa a ejecutar en forma autónoma un programa previamente grabado en la memoria no volátil del MCU (Posición indicada como AUT.)
JE2	La tarjeta se energiza con un eliminador de batería conectado al conector CON1, que proporcione un voltaje de 5 a 7 volts. (Posición denotada como 7.0)	La tarjeta se energiza desde la computadora donde se ejecute el software manejador, esto se logra mediante el módulo IUUM, el cual estaría conectado a la PC vía un puerto USB y a la tarjeta con el conector de 7 hilos H1 * (Posición denotada como 3.3)

* Nota: El módulo IUUM es una Interfaz USB Universal Para Tarjetas MINICON, su conexionado y funcionalidad básica se describen en la sección 14 de esta guía.

TABLA 6. POSTES DE LA TARJETA MINICON_08QG

POSTE	FUNCIONALIDAD
GND (VERTICAL)	CONECTADO A COMÚN DE LA TARJETA (TIERRA)
GND (HORIZONTAL)	CONECTADO A COMUN DE LA TARJETA (TIERRA)
VDD (VERTICAL)	CONECTADO A VDD DE LA TARJETA (+3.3 VOLTS)
VDD (HORIZONTAL)	CONECTADO A VDD DE LA TARJETA (+ 3.3 VOLTS)
RESET/IRQ	CONECTADO AL PIN PTA5/IRQ/TCLK/RESET (3) DEL MCU
5V	EN ESTE POSTE HABRÁ UN VOLTAGE DE 5 VOLTS RESPECTO A TIERRA SOLO CUANDO LA TARJETA SE LIGA A LA PC MEDIANTE EL MÓDULO IUUM

12. Conectores presentes en la tarjeta MINICON_08SH

La tarjeta cuenta cinco conectores. La funcionalidad de éstos se detalla en la tabla 7.

TABLA 7. CONECTORES PRESENTES EN LA TARJETA MINICON_08SH

CONECTOR	FUNCIONALIDAD
CON1	Conector del eliminador de batería empleado para polarizar la tarjeta. El voltaje suministrado por éste debe estar comprendido entre 5 y 7 Volts
CON2	Conector del LCD
CON3	Conector de puertos. Contiene líneas asociados con los bits propios de los puertos del MCU, y además los postes GND y VDD. Véase la figura 41 y la tabla 6
H1	Conector de 7 hilos para ligar la tarjeta con los módulos ISUM o IUUM
POD	Conector para interfaz de hardware (POD) que permite grabar código en el MCU cuando éste no contiene nada de origen en su memoria no volátil

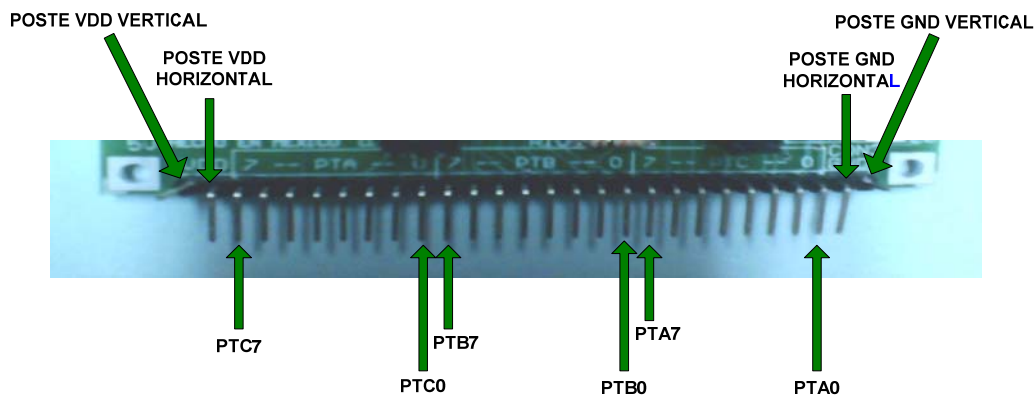


Fig 41. Conector de puertos de la tarjeta MINICON_08SH (CON3)

Para mayor claridad acerca de la funcionalidad de los *jumper*s, postes y conectores de la tarjeta puede verse el diagrama esquemático de ésta contenido en el archivo propio de la referencia [3]. En la figura 42 se muestra la ubicación en la tarjeta de diversos componentes relevantes de ésta.

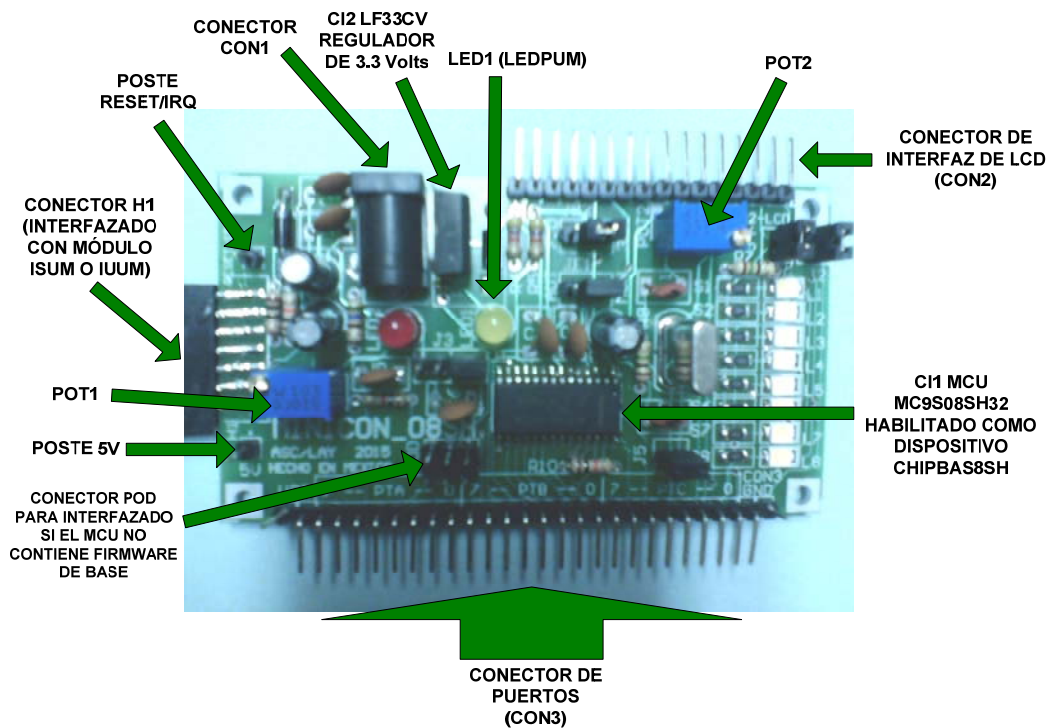


Fig 42. Ubicación de componentes relevantes de la tarjeta MINICON_08QG

13. Funcionalidad de componentes relevantes que no son conectores, postes o chips

En la tabla 8 se detalla la funcionalidad de algunos de los componentes cuya ubicación se muestra en la figura 42.

TABLA 8. FUNCIONALIDAD DE COMPONENTES RELEVANTES DE LA TARJETA MINICON_08SH QUE NO SON CONECTORES, POSTES O CHIPS

COMPONENTE	FUNCIONALIDAD
LEDPUM (LED1)	Parpadea con una cadencia rápida testificando que se pueden efectuar desde el software manejador PUMMA_EST acciones sobre el MCU
POT1	Potenciómetro de 10k que permite aplicar un voltaje comprendido entre cero y 3.3 Volts al pin PTB2/PIB2/SPSCK/ADP6 (18) DEL MCU. Esto cuando el <i>jumper</i> j1 está colocado
POT2	Ajuste de contraste del LCD conectado a la interfaz propia de éste (CON2)

14. Interfazado entre la computadora anfitriona y la tarjeta MINICON_08SH empleando el módulo IUUM

El módulo IUUM es una Interfaz USB Universal para tarjetas MINICON, que puede emplearse en el sistema AIDA08SH como interfaz entre la tarjeta MINICON_08SH y la computadora anfitriona (CA) donde se ejecute el software manejador PUMMA_EST. Esto en lugar del módulo ISUM. IUUM está basado en el chip MCP2200 de MicroChip, que valida un puerto serie virtual mediante un puerto USB de la CA. En la figura 43 se muestra el conexionado de los elementos de un sistema AIDA08SH donde el módulo IUUM es uno de sus componentes funcionales.

COMPUTADORA EJECUTANDO EL
SOFTWARE MANEJADOR
PUMMA_08+ ó PUMMA_EST

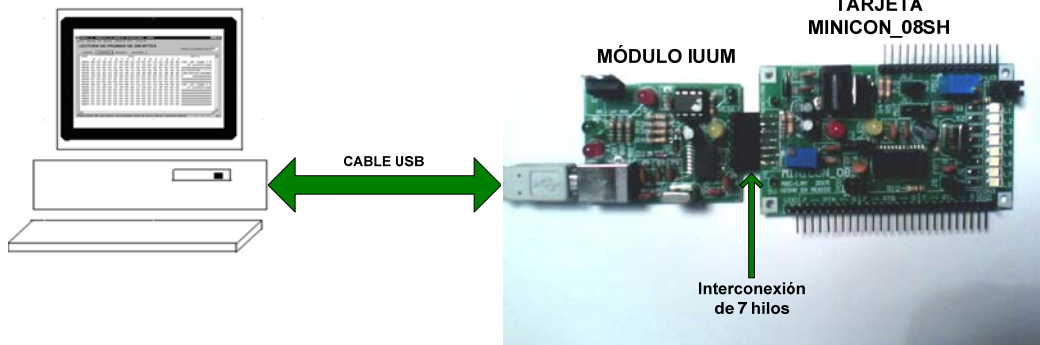


Fig. 43. Sistema AIDA08SH donde se emplea el módulo IUUM

Desde luego que antes de usar un sistema AIDA08SH donde se use un módulo IUUM, se debe instalar el *driver* propio del chip MCP2200 para fines de la validación del puerto USB-SERIE que se realiza con este dispositivo. Los pasos a seguir para esto serían los descritos en la sección 3 de esta guía. El *driver* puede descargarse desde la página de MicroChip en la liga: www.microchip.com/MCP2200. A la fecha de escritura de esta guía, abril de 2016, se ha verificado la funcionalidad del *driver* para WINDOWS XP, 7 y 8.

Cabe señalar además que, cuando se use el módulo IUUM en un sistema AIDA08SH, se tendrán dos alternativas para energizar la tarjeta MINICON_08SH estas son:

- Energización desde la PC donde corre el software manejador. El *jumper* excluyente JE2 deberá estar en la posición (3.3).
- Energización desde un eliminador de batería (5 a 7 Volts), conectado al conector CON1. El *jumper* excluyente JE2 deberá estar en la posición (7.0).

En la figura 44 se muestra el módulo IUUM.

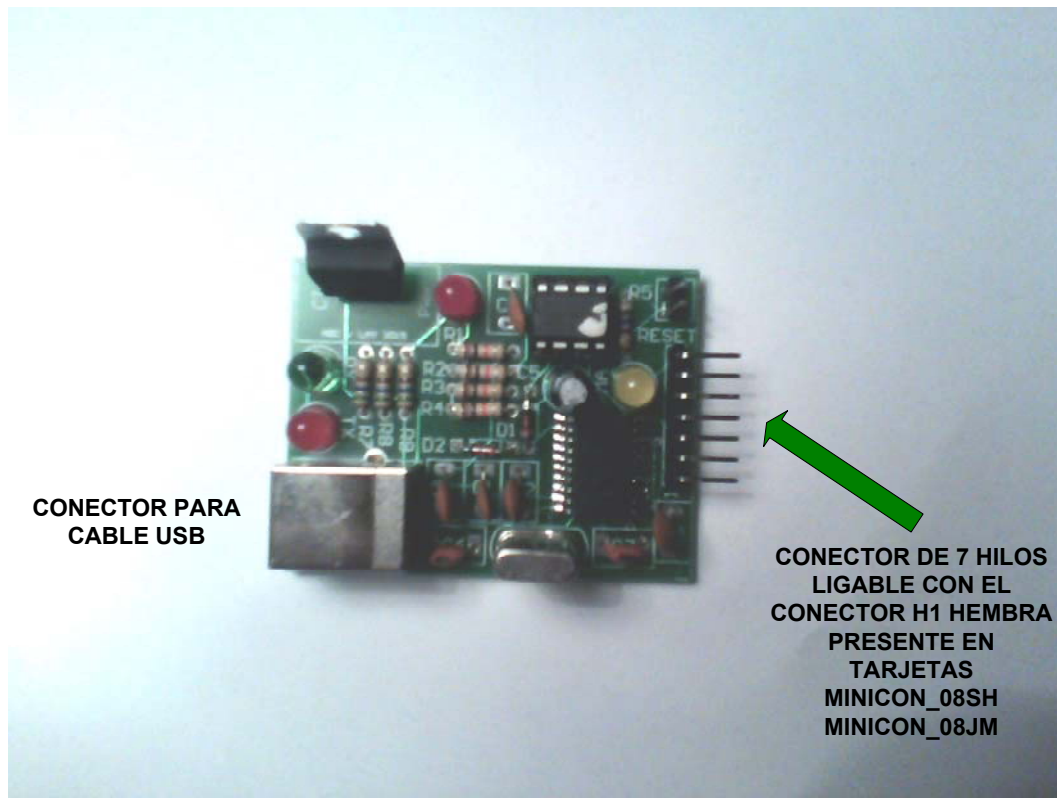


Fig. 44. Módulo IUUM para interfazado entre tarjetas MINICON y computadora anfitriona para fines de un sistema AIDA08SH.

15. Versiones del firmware de base para los dispositivos CHIPBAS8SH

Acorde con la forma en que se generen las señales de reloj del MCU, y la funcionalidad asociada con el pin **PTA5/IRQ/TCLK/RESET** de éste, existen cuatro versiones para el firmware de base que habilita como dispositivo CHIPBAS8SH al microcontrolador MC9S08SH32. A éstas se les denomina respectivamente como NBCP8A_SH32_BIBBAS8, NBCP8B_SH32_BIBBAS8, NBCP8C_SH32_BIBBAS8, NBCP8D_SH32_BIBBAS8. En la tabla 9 se muestran las características básicas acerca de la generación de la señal de reloj de bus y la funcionalidad del pin mencionado para cada una de las versiones del firmware de base aquí mencionadas.

TABLA 9. SEÑAL DE RELOJ DE BUS Y FUNCIONALIDAD DEL PIN PTA5/IRQ/TCLK/RESET, PARA LAS VERSIONES DEL FIRMWARE DE BASE.

Versión de Firmware de base	PTA5/IRQ/TCLK/RESET	Señal de reloj de bus
NBCP8A_SH32_BIBBAS8	Funciona sólo como RESET	Fbus=4Fxtal
NBCP8B_SH32_BIBBAS8	Por default funciona como pta5. Es configurable para funcionar como irq o telk	Fbus=4Fxtal
NBCP8C_SH32_BIBBAS8	Funciona sólo como RESET	Fbus a partir de oscilador interno del MCU
NBCP8D_SH32_BIBBAS8	Por default funciona como pta5. Es configurable para funcionar como irq o telk	Fbus a partir de oscilador interno del MCU

Nota: Fxtal es la frecuencia de un cristal externo conectado a los pines EXTAL y XTAL

Para las cuatro versiones del firmware de base la frecuencia del reloj de bus (fbus) es de 20 MHz, esto hace que se requiera que, para las dos primeras versiones, la frecuencia del cristal externo (Fxtal) deba ser 5 MHz.

15.1 Líneas de puerto y puntos de acceso a periféricos disponibles para el usuario en la tarjeta MINICON_08SH

Debido a características propias de la tarjeta MINICON_08SH y a las variantes de funcionalidad descritas en la tabla 9, el número de líneas de puerto y puntos de acceso a periféricos disponibles para el usuario varía de una a otra versión del firmware de base. En las tablas 10, 11, 12 y 13, para cada caso, se detalla que líneas de puerto están libremente disponibles para el usuario y que uso tienen las líneas no disponibles para el usuario.

TABLA 10. USO DE LÍNEAS DE PUERTO DE LA TARJETA MINICON_08SH PARA LA VERSIÓN NBCP8A_SH32 BIBAS8 DEL FIRMWARE DE BASE

Línea	USO en tarjeta MINICON_08SH
PTA0	Libre uso como: PTA0 ó PIA0 ó TPM1CH0 ó ADP0 ó ACMP+
PTA1	Libre uso como: PTA1 ó PIA1 ó TPM2CH0 ó ADP1 ó ACMP–
PTA2	Libre uso como: PTA2 ó PIA2 ó SDA ó ADP2
PTA3	Libre uso como: PTA3 ó PIA3 ó SCL ó ADP3
PTA4	Libre uso como: PTA4 ó ACMPO ó BKGD ó MS. Ver nota 1
PTA5	Funciona como línea de RESET
PTA6	Libre uso como: PTA6 ó TPM2CH0
PTA7	Bitpum asociado con el led testigo del receptor de comandos
PTB0	Configurado como RxD del puerto serie de enlace con la computadora donde corre el software manejador PUMMA_EST
PTB1	Configurado como TxD del puerto serie de enlace con la computadora donde corre el software manejador PUMMA_EST
PTB2	Libre uso como: PTB2 ó PIB2 ó SPCK ó ADP6
PTB3	Libre uso como: PTB3 ó PIB3 ó MOSI ó ADP7
PTB4	Libre uso como: PTB4 ó TPM2CH1 ó MISO
PTB5	Libre uso como: PTB5 ó TPM1CH1 ó SS
PTB6	Asociado con línea XTAL para conexión de cristal externo
PTB7	Asociado con línea EXTAL para conexión de cristal externo
PTC0	Libre uso como: PTC0 ó TPM1CH0 ó ADP8
PTC1	Libre uso como: PTC1 ó TPM1CH1 ó ADP9
PTC2	Libre uso como: PTC2 ó ADP10
PTC3	Libre uso como: PTC3 ó ADP11
PTC4	Libre uso como: PTC4 ó ADP12
PTC5	Libre uso como: PTC5 ó ADP13
PTC6	Libre uso como: PTC6 ó ADP14
PTC7	Libre uso como: PTC7 ó ADP15

Nota 1: PTA4 funciona únicamente como salida en el MCU MC9S08SH32

TABLA 11. USO DE LÍNEAS DE PUERTO DE LA TARJETA MINICON_08SH PARA LA VERSIÓN NBCP8B_SH32_BIBAS8 DEL FIRMWARE DE BASE

Línea	USO en tarjeta MINICON_08SH
PTA0	Libre uso como: PTA0 ó PIA0 ó TPM1CH0 ó ADP0 ó ACMP+
PTA1	Libre uso como: PTA1 ó PIA1 ó TPM2CH0 ó ADP1 ó ACMP–
PTA2	Libre uso como: PTA2 ó PIA2 ó SDA ó ADP2
PTA3	Libre uso como: PTA3 ó PIA3 ó SCL ó ADP3
PTA4	Libre uso como: PTA4 ó ACMPO ó BKGD ó MS. Ver nota 1
PTA5	Libre uso como: PTA5 ó IRQ ó TCLK
PTA6	Libre uso como: PTA6 ó TPM2CH0
PTA7	Bitpum asociado con el led testigo del receptor de comandos
PTB0	Configurado como RxD del puerto serie de enlace con la computadora donde corre el software manejador PUMMA EST
PTB1	Configurado como TxD del puerto serie de enlace con la computadora donde corre el software manejador PUMMA EST
PTB2	Libre uso como: PTB2 ó PIB2 ó SPSCCK ó ADP6
PTB3	Libre uso como: PTB3 ó PIB3 ó MOSI óADP7
PTB4	Libre uso como: PTB4 ó TPM2CH1 ó MISO
PTB5	Libre uso como: PTB5 ó TPM1CH1 ó SS
PTB6	Asociado con línea XTAL para conexión de cristal externo
PTB7	Asociado con línea EXTAL para conexión de cristal externo
PTC0	Libre uso como: PTC0 ó TPM1CH0 ó ADP8
PTC1	Libre uso como: PTC1 ó TPM1CH1 ó ADP9
PTC2	Libre uso como: PTC2 ó ADP10
PTC3	Libre uso como: PTC3 ó ADP11
PTC4	Libre uso como: PTC4 ó ADP12
PTC5	Libre uso como: PTC5 ó ADP13
PTC6	Libre uso como: PTC6 ó ADP14
PTC7	Libre uso como: PTC7 ó ADP15

Nota 1: PTA4 funciona únicamente como salida en el MCU MC9S08SH32

TABLA 12. USO DE LÍNEAS DE PUERTO DE LA TARJETA MINICON_08SH PARA LA VERSIÓN NBCP8C_SH32_BIBAS8 DEL FIRMWARE DE BASE

Línea	USO en tarjeta MINICON_08SH
PTA0	Libre uso como: PTA0 ó PIA0 ó TPM1CH0 ó ADP0 ó ACMP+
PTA1	Libre uso como: PTA1 ó PIA1 ó TPM2CH0 ó ADP1 ó ACMP–
PTA2	Libre uso como: PTA2 ó PIA2 ó SDA ó ADP2
PTA3	Libre uso como: PTA3 ó PIA3 ó SCL ó ADP3
PTA4	Libre uso como: PTA4 ó ACMPO ó BKGD ó MS. Ver nota 1
PTA5	Funciona como línea de RESET
PTA6	Libre uso como: PTA6 ó TPM2CH0
PTA7	Bitpum asociado con el led testigo del receptor de comandos
PTB0	Configurado como RxD del puerto serie de enlace con la computadora donde corre el software manejador PUMMA EST
PTB1	Configurado como TxD del puerto serie de enlace con la computadora donde corre el software manejador PUMMA EST
PTB2	Libre uso como: PTB2 ó PIB2 ó SPSCCK ó ADP6
PTB3	Libre uso como: PTB3 ó PIB3 ó MOSI óADP7

TABLA 12. (CONTINUACIÓN)

PTB4	Libre uso como: PTB4 ó TPM2CH1 ó MISO
PTB5	Libre uso como: PTB5 ó TPM1CH1 ó SS
PTB6	Libre uso como: PTB6 ó SDA
PTB7	Libre uso como: PTB7 ó SCL
PTC0	Libre uso como: PTC0 ó TPM1CH0 ó ADP8
PTC1	Libre uso como: PTC1 ó TPM1CH1 ó ADP9
PTC2	Libre uso como: PTC2 ó ADP10
PTC3	Libre uso como: PTC3 ó ADP11
PTC4	Libre uso como: PTC4 ó ADP12
PTC5	Libre uso como: PTC5 ó ADP13
PTC6	Libre uso como: PTC6 ó ADP14
PTC7	Libre uso como: PTC7 ó ADP15

Nota 1: PTA4 funciona únicamente como salida en el MCU MC9S08SH32

TABLA 13. USO DE LÍNEAS DE PUERTO DE LA TARJETA MINICON_08SH PARA LA VERSIÓN NBCP8D_SH32 BIBAS8 DEL FIRMWARE DE BASE

Línea	USO en tarjeta MINICON_08SH
PTA0	Libre uso como: PTA0 ó PIA0 ó TPM1CH0 ó ADP0 ó ACMP+
PTA1	Libre uso como: PTA1 ó PIA1 ó TPM2CH0 ó ADP1 ó ACMP-
PTA2	Libre uso como: PTA2 ó PIA2 ó SDA ó ADP2
PTA3	Libre uso como: PTA3 ó PIA3 ó SCL ó ADP3
PTA4	Libre uso como: PTA4 ó ACMPO ó BKGD ó MS. Ver nota 1
PTA5	Libre uso como: PTA5 ó IRQ ó TCLK
PTA6	Libre uso como: PTA6 ó TPM2CH0
PTA7	Bitpump asociado con el led testigo del receptor de comandos
PTB0	Configurado como RxD del puerto serie de enlace con la computadora donde corre el software manejador PUMMA_EST
PTB1	Configurado como TxD del puerto serie de enlace con la computadora donde corre el software manejador PUMMA_EST
PTB2	Libre uso como: PTB2 ó PIB2 ó SPCK ó ADP6
PTB3	Libre uso como: PTB3 ó PIB3 ó MOSI ó ADP7
PTB4	Libre uso como: PTB4 ó TPM2CH1 ó MISO
PTB5	Libre uso como: PTB5 ó TPM1CH1 ó SS
PTB6	Libre uso como: PTB6 ó SDA
PTB7	Libre uso como: PTB7 ó SCL
PTC0	Libre uso como: PTC0 ó TPM1CH0 ó ADP8
PTC1	Libre uso como: PTC1 ó TPM1CH1 ó ADP9
PTC2	Libre uso como: PTC2 ó ADP10
PTC3	Libre uso como: PTC3 ó ADP11
PTC4	Libre uso como: PTC4 ó ADP12
PTC5	Libre uso como: PTC5 ó ADP13
PTC6	Libre uso como: PTC6 ó ADP14
PTC7	Libre uso como: PTC7 ó ADP15

Nota 1: PTA4 funciona únicamente como salida en el MCU MC9S08SH32

15.2 Testificación al RESET de la versión del firmware de base presente

Cuando el Jumper excluyente JE1 está en la posición MON, la versión del firmware de base se testifica al RESET, mediante un determinado número de parpadeos del bitpum y la presencia de un número en los leds testigo del estado lógico de los bits del puerto C (ptcd) del MCU. Véase la tabla 14.

TABLA 14. TESTIFICACIÓN DEL FIRMWARE DE BASE PRESENTE DESPUÉS DE UN RESET CUANDO JE1 ESTÁ EN LA POSICIÓN MON

Versión de Firmware de base	Parpadeos de bitpum	Número en puerto C
NBCP8A SH32 BIBBAS8	No hay parpadeos	0
NBCP8B SH32 BIBBAS8	1	1
NBCP8C SH32 BIBBAS8	2	2
NBCP8D SH32 BIBBAS8	3	3

15. Referencias

- [1] Salvá, A (2011). *MANUAL DE USUARIO DEL SISTEMA AIDA08*. Archivo `muaida08ve2011.pdf`, descargable desde <http://dctrl.fi-b.unam.mx/~salva>
- [2] FREESCALE (2008), MC9S08SH32/16. Data sheet contenido en el archivo `mc9s08sh32.pdf`
- [3] Salvá A y Altamirano L (2012). Diagrama de la tarjeta MINICON_08SH, contenido en el archivo `MINICON_08SH_14OCT2015_Diagrama.pdf`
- [4] Salvá, A y L. Altamirano (2009). DISPOSITIVOS CHIPBAS8, MICROCONTROLADORES HC08 PROGRAMABLES EN LENGUAJE BASIC. Memoria del Simposio Anual de Automatización, Electrónica e Instrumentación, (SAAEI 2009). Celebrado en julio de 2009 en la Universidad Carlos III en Madrid España. El artículo puede verse en el archivo `chipbas8_art_saaei09_def.pdf` descargable desde <http://dctrl.fi-b.unam.mx/~salva>.
- [5] Salvá, A (2016). Guía Básica de Usuario para el desarrollo de programas bajo el compilador cruzado SBAS8. Archivo `gbu_sbas8.pdf`.