

SAFE Bootloader

Nº Revisión	Realizó	Revisó	Aprobó	Fecha Emisión	Descripción
1	Leandro G. Francucci	Leandro G. Francucci	Leandro G. Francucci	19/03/08	

Contenido

Contenido.....	2
Lista de figuras.....	2
Lista de tablas.....	3
Objetivos.....	4
Mapa de memoria del MCU target.....	5
Bootloader residente.....	8
Procesamiento del archivo s19.....	23
Port del preprocesador s19.....	32
Conversor S19.....	32

Lista de figuras

Figura 1. Mapa de memoria del MCU GT/GB60.....	5
Figura 6. Diagrama lógico de compare_versions.....	12
Figura 5. Diagrama lógico de check_file.....	13
Figura 9. Diagrama lógico de init_page.....	14
Figura 11. Diagrama lógico de fgetc.....	15
Figura 11. Diagrama lógico de spi_select y spi_deselect.....	16
Figura 10. Diagrama lógico de read_page.....	17
Figura 10. Diagrama lógico de send_command.....	18
Figura 10. Diagrama lógico de xfer_page_ram.....	19
Figura 7. Diagrama lógico de download_file.....	20
Figura 3. Diagrama lógico de check_usr_code.....	21
Figura 12. Reubicación de la tabla de vectores.....	26

Lista de tablas

Tabla 1. Port bootloader residente MCU.....	22
Tabla 2. Conversor S19.....	32
Tabla 3. Archivos de salida.....	33

Objetivos

El bootloader permite la reprogramación *in-circuit* de un MCU de la familia GT/GB60A(non-A) en la aplicación CIM. A continuación se listan sus objetivos:

1. reprogramación *in-circuit* del MCU (etapa primaria y secundaria) soportado por la aplicación CIM, placa SAFEMAIN.
2. Protección contra sobre-escrituras y borrado no intencional de la zona de memoria asignada al bootloader.
3. Lectura de imagen de aplicación de usuario o CIM desde memoria DataFlash instalada.
4. Determina si debe descargar la imagen almacenada en DataFlash en cada *boot* del sistema.
5. Descarga la imagen de la aplicación de usuario o CIM desde la memoria DataFlash instalada, de acuerdo a la etapa en funcionamiento.
6. Ejecuta la aplicación de usuario o CIM.
7. Ocupación mínima de espacio en FLASH del MCU.

Mapa de memoria del MCU target

El mapa de memoria que se muestra a continuación pertenece al HCS08 GT/GB60A(non-A), en donde se detalla cada una de sus particiones.

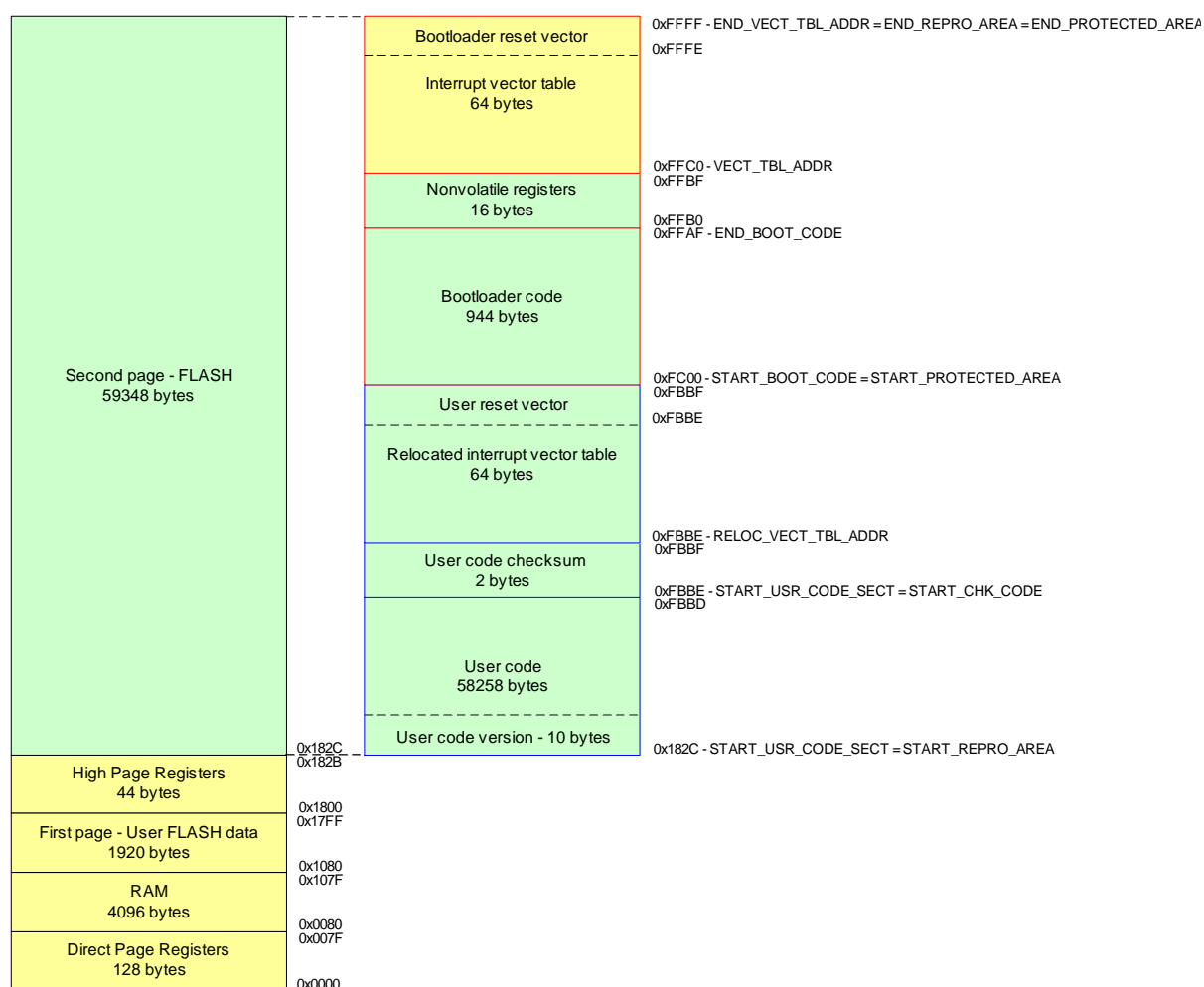


Figura 1. Mapa de memoria del MCU GT/GB60

Aquellos sectores de memoria rellenos en color amarillo están impuestos por el MCU, los rellenos en verde son particiones creadas para realizar la reprogramación *in-circuit*, los sectores marcados en contorno rojo pertenecen al área protegida, mientras que los marcados en contorno azul pertenecen al área reprogramable.

Si la aplicación requiere almacenamiento de datos no-volátiles **debe** utilizarse el segmento comprendido entre `START_FIRST_PAGE_ADDR` y `END_FIRST_PAGE_ADDR`, ya que permanece invisible al bootloader y al procesador de archivos s19.

La tabla de vectores está compuesta por 32 vectores, incluyendo el vector de reset, ubicado en sus dos últimas direcciones, generando una zona de 64 bytes.

Si se protege la zona de memoria que contiene el bootloader y su tabla de vectores de interrupción, el MCU asume que la misma será reubicada. Esta protección implica que el bootloader residente y sus vectores de interrupción no podrán ser sobrescritos ni borrados por la aplicación de usuario. El segmento de memoria comprendido entre `START_PROTECTED_AREA` y `END_PROTECTED_AREA` es el protegido, luego, el MCU reubica la tabla de vectores de interrupción a la posición de memoria `START_PROTECTED_AREA-1`. El vector de reset es reubicable, y por lo tanto, luego de un reset del MCU, la dirección de salto en reset permanecerá en la posición `0xFFFFE:FFFF`, mientras que la posición del vector de reset de la aplicación se ubicará en la dirección `START_PROTECTED_AREA-2:START_PROTECTED_AREA-1`. Luego, la tabla de vectores se ubica a partir de la dirección `RELOC_VECT_TBL_ADDR`.

El segmento de memoria comprendido entre la dirección `START_USR_CODE_SECT` y `START_BOOT_CODE-1`, es el reprogramable por el bootloader. Este contiene el bloque de memoria en donde reside la aplicación de usuario, su tabla de vectores de interrupción y su checksum. La dirección `START_CHK_CODE:START_CHK_CODE+1` contiene el checksum del programa de aplicación. Este se calcula como la suma en 16-bits complemento a 1, donde el byte más significativo se encuentra en la dirección `START_CHK_CODE`.

A partir de la dirección `START_USR_CODE_SECT` se encuentra la cadena de versión de la aplicación y del hardware, codificada en ASCII terminada en NULL, bajo el siguiente formato: "SWxx.yy.zzHWss.gg\0".

Cuando se genera un reset, el MCU toma la dirección de salto desde la dirección `0xFFFFE:FFFF`, vector de reset, en donde reside la dirección de comienzo del bootloader. Luego, si el bootloader decide ejecutar la aplicación, su dirección de comienzo se

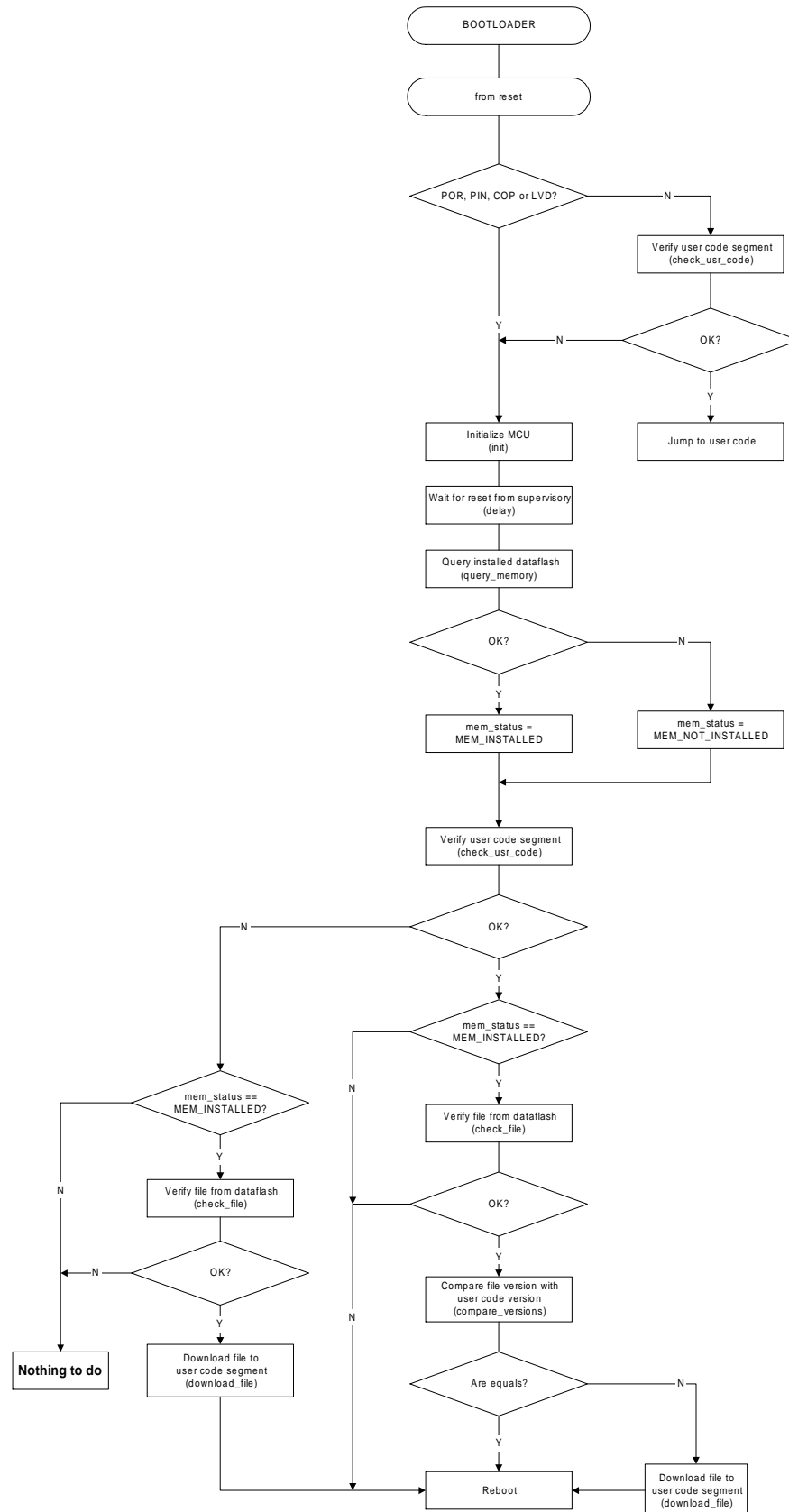
encuentra en la dirección `START_PROTECTED_AREA-2:START_PROTECTED_AREA-1`, siendo este el vector de reset original de la aplicación.

Bootloader residente

Primeramente el bootloader determina la fuente del reset, si la misma es POR, PIN, COP o LVD continua su ejecución para luego determinar si debe o no, actualizar la memoria FLASH del MCU. Por otro lado, si la fuente de reset es ILOP o IA, determina si existe una aplicación de usuario válida en memoria FLASH del MCU, si es así, la ejecuta, de lo contrario continua su ejecución al igual que con las fuentes anteriores de reset.

Para determinar si existe programa almacenado en FLASH del MCU, verifica que el vector de reset de la aplicación contenga una dirección de salto válida y calcula el checksum de la aplicación almacenada en FLASH del MCU. Para ello, el bootloader conoce la dirección del vector de reset de la aplicación de usuario o `USR_RESET_VECT:USR_RESET_VECT+1`, los límites del bloque de memoria a verificar, determinados por `START_USR_CODE_SECT` a `END_USR_CODE_SECT` y la dirección en donde reside el checksum de la aplicación de usuario, `START_CHK_CODE:START_CHK_CODE+1`, en donde el byte más significativo de la suma complemento se encuentra en `START_CHK_CODE`.

El checksum se calcula como la suma en 16-bits complemento a 1, del contenido del segmento `USR_CODE_SECT`. Para ejecutar la aplicación de usuario, determina la dirección de inicio desde `USR_RESET_VECT:USR_RESET_VECT+1`.



Luego, verifica si existe una aplicación válida en FLASH del MCU, si es así y a su vez existe una memoria instalada compatible con las esperadas, verifica que dentro de esta última se encuentre el archivo **s08imdwr**, para ello calcula el checksum (en 16-bits complemento a 1) del archivo completo, luego si la suma calculada coincide con la almacenada dentro del archivo procede a determinar si debe realizar la reprogramación de FLASH del MCU con lo almacenado en la memoria DataFlash. Esto último se realiza, si y solo si, la cadena de versión de la aplicación de usuario, almacenada en FLASH del MCU, es diferente a la cadena de versión del archivo **s08imdwr**. Si las cadenas difieren, implica que debe reprogramarse la FLASH del MCU con la aplicación que reside en la memoria DataFlash. Caso contrario, realiza un *reboot* para ejecutar la aplicación de usuario almacenada en FLASH del MCU.

El archivo **s08imdwr** se encuentra a partir de la página PAGE_OFFSET de la memoria DataFlash y su tamaño está determinado por FILE_SIZE. Mientras que el checksum del mismo se encuentra en sus dos primeras posiciones, siendo la primera de ellas el byte más significativo. Luego, la cadena de versión del archivo se encuentra en la posición START_FILE_VERSION cuya longitud es VERSION_STRING_SIZE.

La versión de la aplicación de usuario almacenada en FLASH del MCU se encuentra desde la dirección START_USR_CODE_SECT y su longitud impuesta por VERSION_STRING_SIZE.

Para reprogramar la FLASH del MCU con una versión diferente de aplicación de usuario, el bootloader recorre y analiza el contenido del archivo **s08imdwr** a partir de la posición START_FILE_ADDR y ejecuta los comandos del protocolo FC que lo componen. Estos pueden ser, programar datos en FLASH del MCU, borrar un sector de FLASH del MCU o generar un *reboot*. Este último indica el final del archivo.

Para recorrer el archivo **s08imdwr** el bootloader lee las páginas de la memoria DataFlash que se requieran de acuerdo con FILE_SIZE y las almacena en RAM del MCU, para su posterior utilización.

Leer una página de la memoria DataFlash implica utilizar el comando MM_PAGE_READ y su algoritmo, el cual implica enviar el comando, luego los 20 bits de la dirección de

lectura requerida y 32 bits sin importar su contenido. Luego, BIN_PAGE_SIZE lecturas para transferir el contenido de la página requerida a RAM del MCU. Ver documento *AN-4 Using Atmel's DataFlash*.

En caso que no exista aplicación de usuario válida en FLASH del MCU y exista una memoria instalada compatible, verifica si existe un archivo **s08imdwr** válido, si es así, lo descarga a FLASH del MCU.

A continuación se listan cada uno de los diagramas del bootloader.

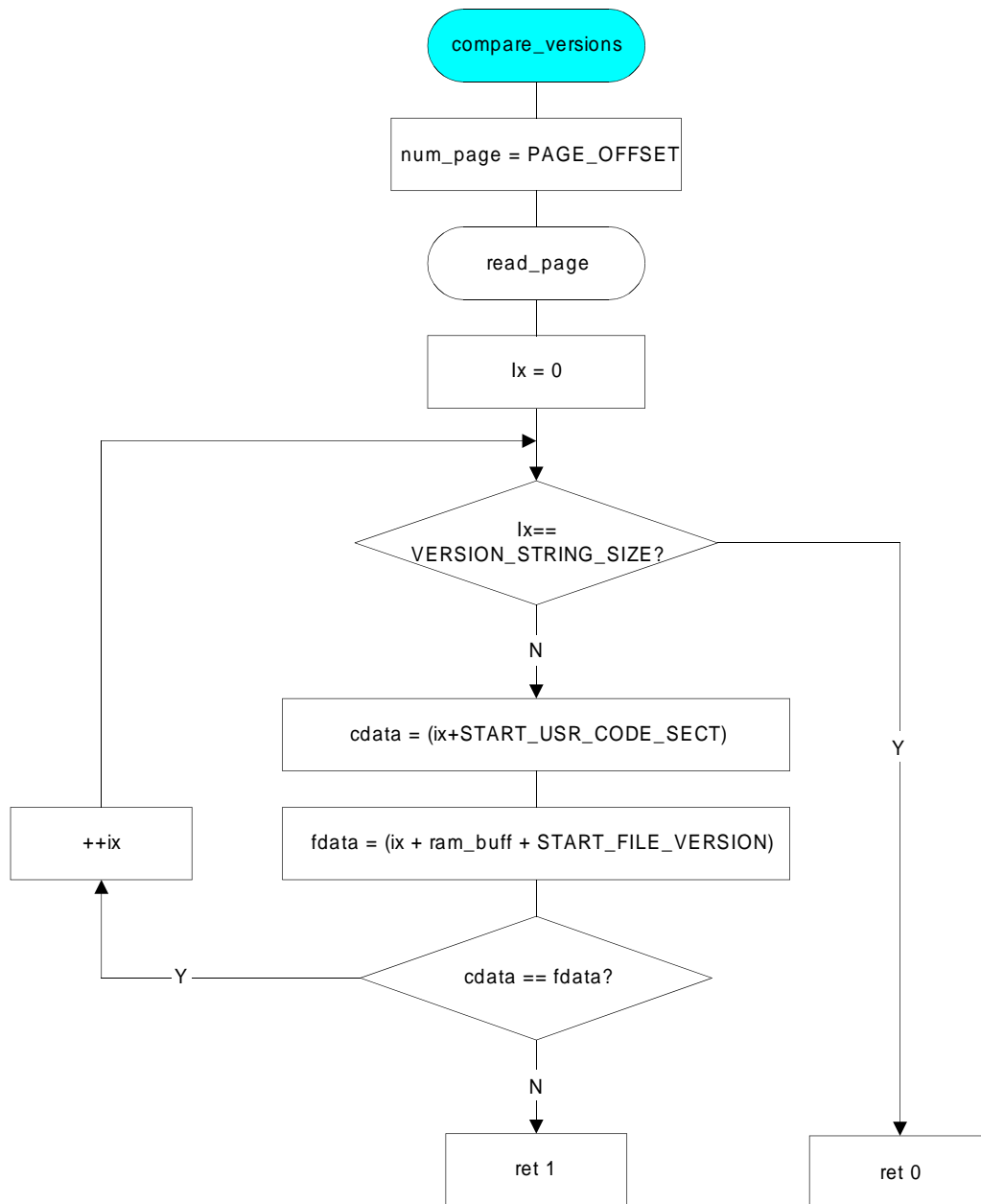


Figura 6. Diagrama lógico de *compare_versions*

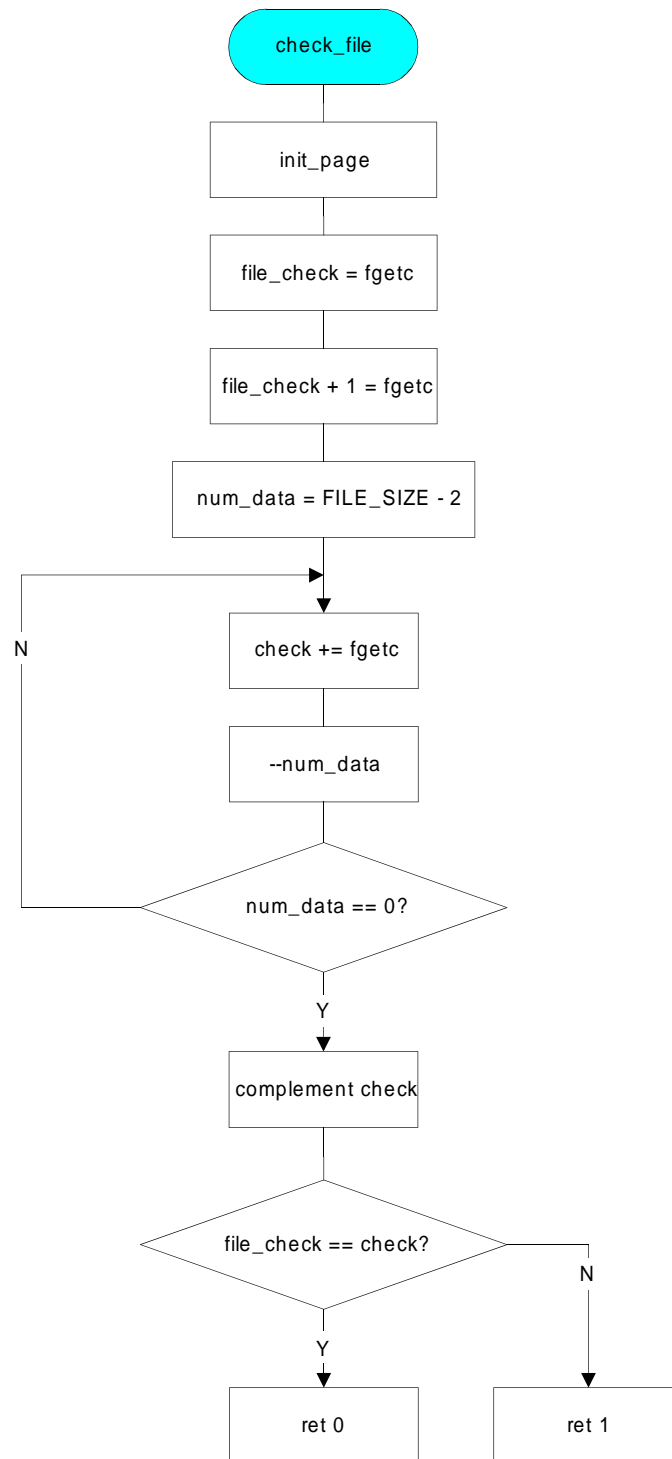


Figura 5. Diagrama lógico de *check_file*

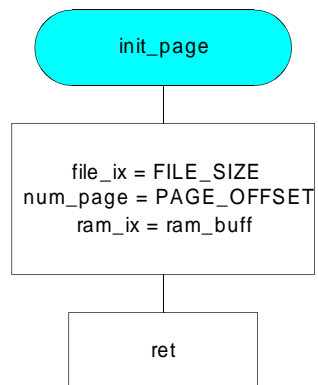


Figura 9. Diagrama lógico de *init_page*

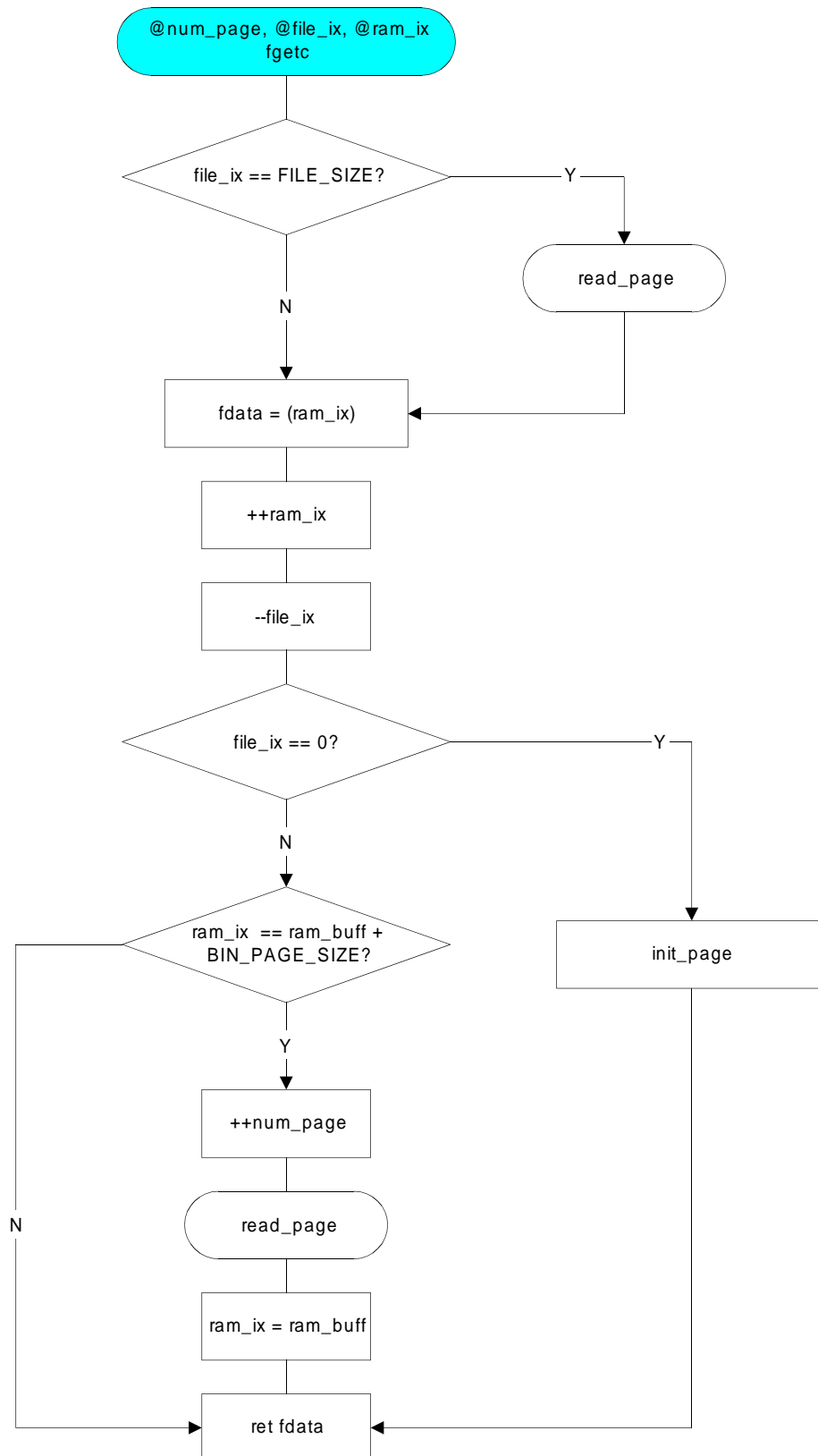


Figura 11. Diagrama lógico de *fgetc*

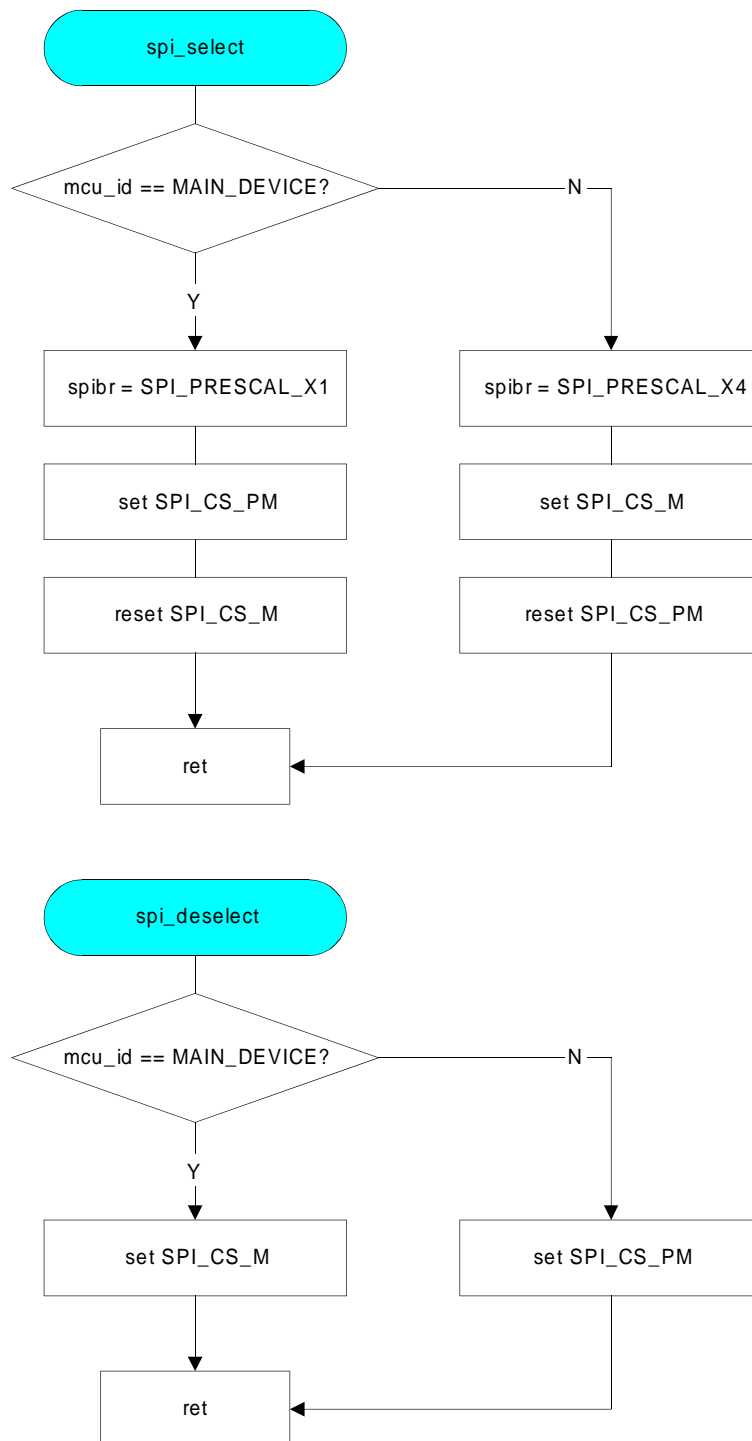


Figura 11. Diagrama lógico de *spi_select* y *spi_deselect*

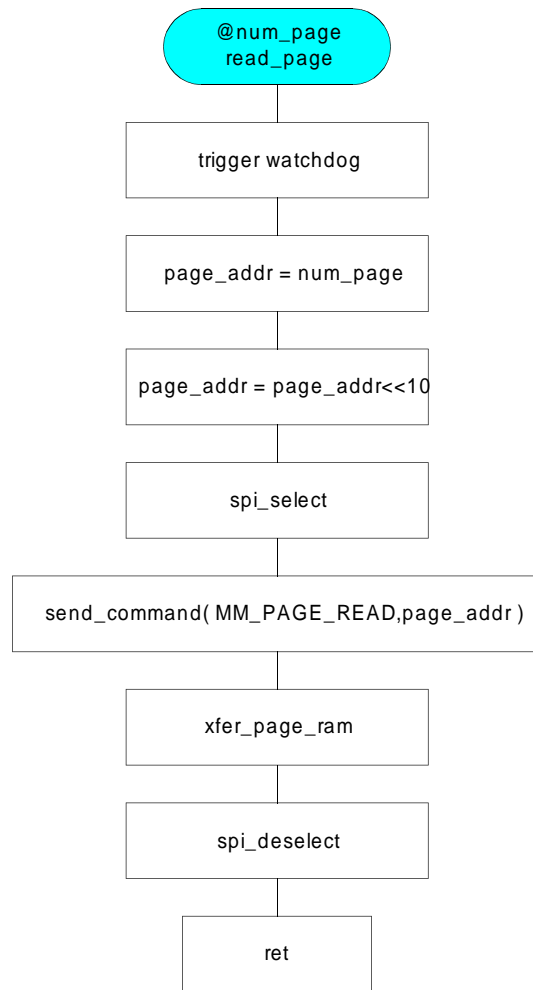


Figura 10. Diagrama lógico de *read_page*

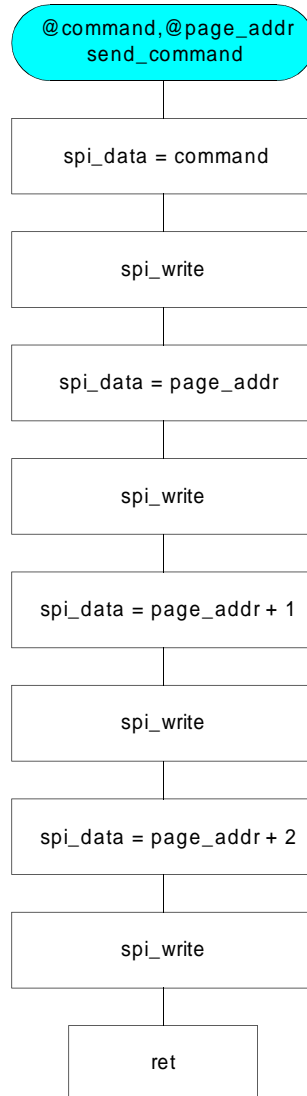


Figura 10. Diagrama lógico de *send_command*

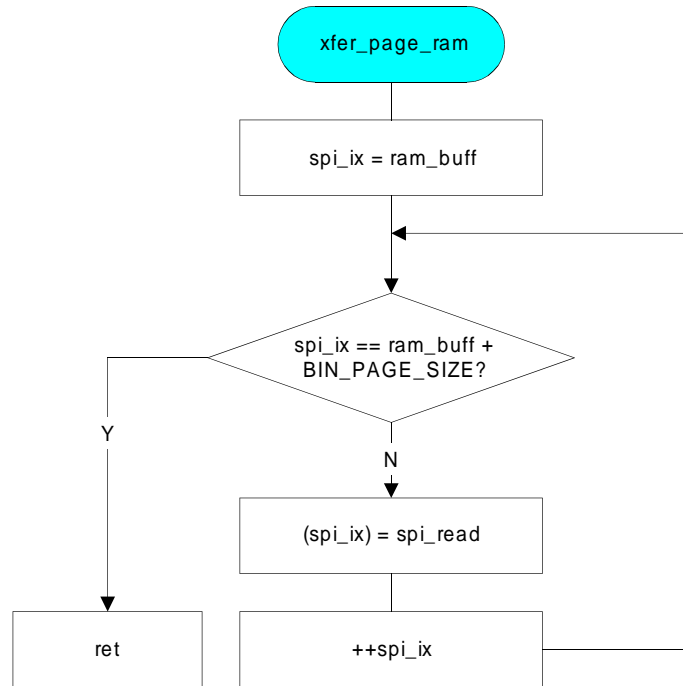


Figura 10. Diagrama lógico de *xfer_page_ram*

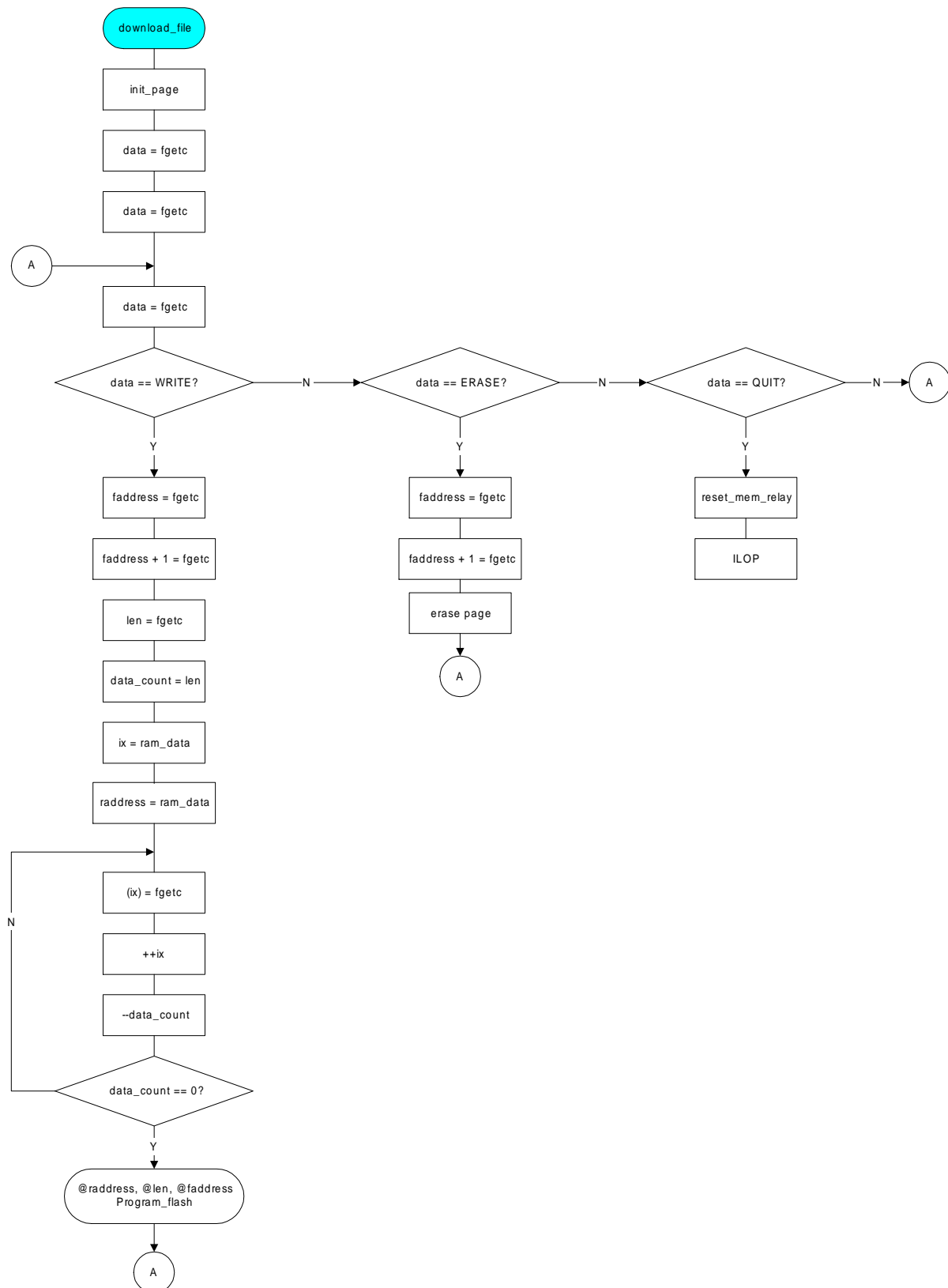


Figura 7. Diagrama lógico de *download_file*

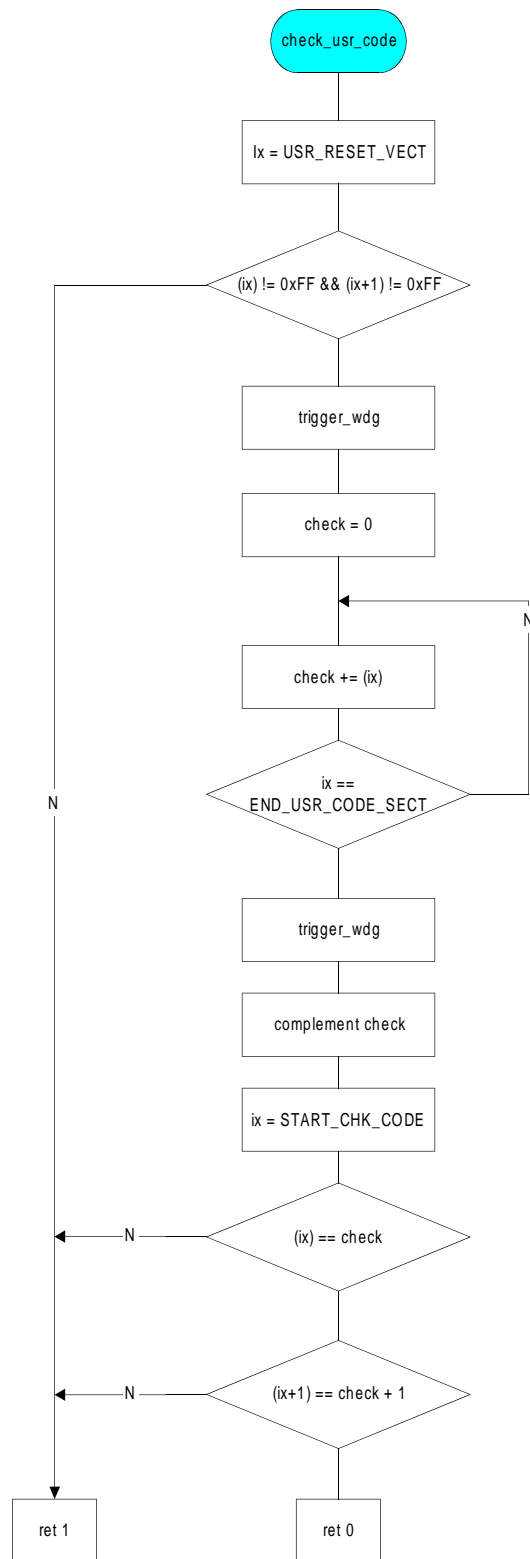


Figura 3. Diagrama lógico de *check_usr_code*

La siguiente tabla describe y detalla cada una de las constantes que deben definirse en el bootloader. El mismo criterio debe utilizarse en el software que genera la imagen **s08imdwr**.

Constantes	Descripción	Gx60A(non-A)
FILE_SIZE	Tamaño del archivo s08imdwr almacenado en la memoria DataFlash	62317
BIN_PAGE_SIZE	Tamaño de la página efectiva de la memoria DataFlash	512
START_FILE_VERSION	Posición de comienzo de la cadena de versión de la aplicación de usuario	9
VERSION_STRING_SIZE	Tamaño de la cadena de versión de la aplicación de usuario	10
ERBLK_LEN	Tamaño del sector de borrado de FLASH	512
WRBLK_LEN	Tamaño del sector de escritura de FLASH	128
START_USR_CODE_SECT	Dirección de inicio de código de usuario	0x182C
START_REPRO_AREA	Dirección de inicio de la zona de memoria FLASH reprogramable	START_USR_CODE_SECT
END_USR_CODE_SECT	Dirección de fin del código de usuario	0xFBFE
START_CHK_CODE	Dirección de inicio del checksum del código de usuario	END_USR_CODE_SECT
RELOC_VECT_TBL_ADDR	Dirección de inicio de la tabla de vectores reubicada	0xFBC0
START_BOOT_CODE	Dirección de inicio del código del bootloader residente	0xFC00
VECT_TBL_ADDR	Dirección de inicio de la tabla de vectores original	0xFFC0
END_VECT_TBL_ADDR	Dirección de fin de la tabla de vectores original	0xFFFF
END_REPRO_AREA	Dirección de fin de la zona de memoria FLASH reprogramable	END_VECT_TBL_ADDR
RESET_VECTOR	Dirección del vector de reset original	END_VECT_TBL_ADDR - 1
USR_RESET_VECT	Dirección del vector de reset de la aplicación de usuario	START_BOOT_CODE - 2
PAGE_OFFSET	Número de página en la cual comienza el archivo s08imdwr en la memoria DataFlash	216

Tabla 1. Port bootloader residente MCU

Procesamiento del archivo s19

El procesamiento de archivos s19 depende exclusivamente del mapa de memoria que posea el microcontrolador a utilizar.

El funcionamiento del procesador de archivos s19 es directo, siendo una simple secuencia de pasos:

1. Abre el archivo s19 en cuestión.
2. Procesa el s19 extrayendo y almacenando la información necesaria de cada registro. Esta última se almacena acordemente en un archivo binario llamado **image**, siendo este una imagen de la memoria del MCU, de forma tal que cada dirección de memoria pueda accederse tanto para escritura como lectura. Así, al finalizar el procesamiento, el archivo **image** refleja la memoria interna del MCU luego de la descarga. Inicialmente, todas las posiciones de memoria de **image** se inicializan con el mismo valor (0xFF) que una típica memoria FLASH.

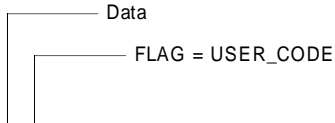
Cada registro de **image** define una posición de memoria direccionable. La siguiente figura muestra el tipo de estructura definida para cada registro.

```
enum
{
    EMPTY, USER_CODE, SYSTEM_CODE,
    NUM_FLAGS
};

typedef struct
{
    unsigned char d; /* data */
    unsigned char f; /* valid flag 0=empty; 1=usercode; 2=systemcode */
} MEM_POS_T;
```

Esto permite direccionar **image**, escribir o leer una posición determinada y verificar si su contenido es válido, está vacío o es utilizable solo por el MCU.

La siguiente figura muestra del archivo **image** un bloque de direcciones perteneciente al bloque de código del usuario.



8b01	8901	9e01	fe01	0501	f601	af01	0101
9e01	ff01	0501	8801	8a01	8101	a701	fc01
c601	8001	8501	4c01	9501	e701	0101	c601
8001	8401	4c01	f701	3201	8001	8601	2001
1f01	8901	8b01	f601	8701	e601	0201	4c01
9e01	e701	0601	e601	0301	ee01	0101	8a01
4c01	2001	0301	7f01	af01	0101	4b01	fb01
9e01	6b01	0501	f701	8a01	8801	af01	0401
9e01	6b01	0201	dd01	9e01	6b01	0101	d901
3201	8001	8801	8901	8b01	ad01	b101	9701
4c01	9e01	e701	0301	ad01	aa01	4c01	9e01

A medida que se identifican los registros del s19 se almacenan en forma descriptiva en el archivo **reclog**. La siguiente figura muestra un fragmento de este archivo, en donde cada registro se identifica con un número consecutivo. La información de cada registro está compuesta por la cantidad de datos en formato decimal codificado en ASCII, la dirección a partir de la cual se almacenarán estos datos en formato hexadecimal codificado en ASCII y finalmente los datos en igual formato.

Records from s19 file - Mon Mar 12 16:38:17 2007

|rec nr: 001 | Length : 032 | Address : 8000|

|D : 8B899EFE05F6AF019EFF05888A81A7FCC680854C95E701C680844CF732808620|

|rec nr: 002 | Length : 032 | Address : 8020|

|D : 1F898BF687E6024C9EE706E603EE018A4C20037FAF014BFB9E6B05F78A88AF04|

...

|rec nr: 075 | Length : 004 | Address : FFDA|

|D : 97DB97C4|

|rec nr: 076 | Length : 002 | Address : FFE2|

|D : 984B|

|rec nr: 077 | Length : 002 | Address : FFEE|

|D : 9846|

|rec nr: 078 | Length : 002 | Address : FFEE|

|D : 807B|

La creación del archivo **reclog** es opcional a través de la constante global **DEBUG**.

También se genera la imagen binaria acorde al archivo original s19, en un archivo denominado **binlog**. Este último es útil para comparar la imagen a cargar con la realmente cargada en el MCU.

3. Reubica la tabla de vectores del s19 original. Esto requiere transferir los vectores de interrupción desde el s19 hacia la zona de vectores reubicados.

La zona marcada en color rojo indica el área de memoria protegida, desde la dirección **START_PROTECTED_AREA** hasta el final de la memoria del MCU o

END_PROTECTED_AREA. Por lo tanto, la tabla de vectores de interrupción se reubica en START_PROTECTED_AREA-1.

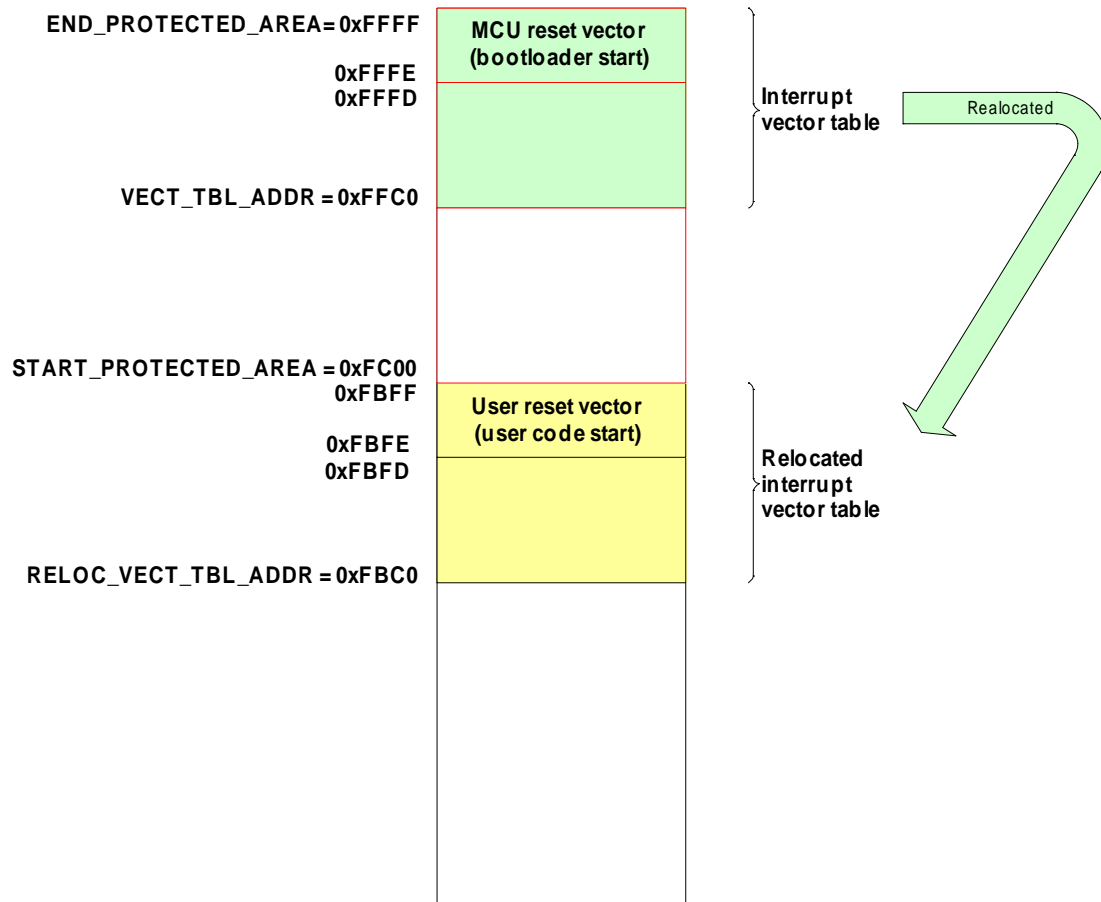


Figura 12. Reubicación de la tabla de vectores

- Verifica que la imagen s19 se encuentre dentro del rango válido de memoria de acuerdo con el MCU elegido. Para ello se barre el archivo **image** desde la dirección inicial 0x0 a la dirección final del mapa de memoria, `END_REPRO_AREA` y se determina que solo exista contenido dentro del bloque de memoria desde `START_REPRO_AREA` a `END_REPRO_AREA`.
- Verifica que la imagen s19 no utilice sectores que correspondan a las áreas protegidas. Para ello barre el archivo **image** desde la dirección `START_BOOT_CODE` a la dirección final del mapa de memoria `END_REPRO_AREA` verificando que no exista contenido válido.

6. Verifica que la imagen s19 no utilice la zona destinada para el checksum del código de usuario ubicada en el rango `START_CHK_CODE:START_CHK_CODE+1`.
7. Verifica si la imagen s19 utiliza la primera página de la memoria FLASH. Si así fuera, el procesador s19 no incluye este contenido dentro de la imagen a descargar, ya que se preve que esta zona sea utilizada para el almacenamiento de datos no-volátiles por el programa de usuario. Este bloque corresponde desde la dirección `START_FIRST_PAGE_ADDR` a `END_FIRST_PAGE_ADDR`.
8. Verifica si la imagen s19 utiliza la zona High Page Registers de la memoria FLASH. Si así fuera, el procesador s19 no incluye este contenido dentro de la imagen a descargar, ya que se preve que esta zona sea utilizada por el programa de usuario. Este bloque corresponde desde la dirección `START_HIGH_PAGE_ADDR` a `END_HIGH_PAGE_ADDR`.
9. Escribe con el valor `EMPTY_FLASH_VALUE` el contenido de aquellas direcciones de memoria, dentro de la partición de código de usuario, que esten vacías. Para ello se barre el archivo **image** desde la dirección `START_USR_CODE_SECT` a la dirección final de este segmento `START_USR_CODE_SECT`. Luego, las direcciones identificadas como vacías se marcan como válidas.

User code end

3201	e601	4001	e601	4f01	e601	5d01	e601
6601	e601	7601	e701	4901	e701	5301	e701
5a01	0001	0401	0201	0001	0001	0001	0001
0001	0001	0001	0001	0001	0001	0001	0001
0001	0001	0001	0001	0001	0001	0001	0001
0001	0001	0001	0001	0001	0001	0001	0001
0001	0001	0001	0001	0001	0001	0001	0001
0001	0001	0001	0001	0001	0001	0001	0001
0001	0001	0001	0001	0001	0001	0001	0001
0001	0001	0001	0001	0001	0001	0001	0001
0001	0001	0001	0001	0001	0001	0001	0001
0001	0001	0001	0001	0001	0001	0001	0001
0001	0001	0001	0001	0001	0001	0001	0001
0001	0001	0001	0001	0001	0001	0001	0001
0001	0001	0001	0001	0001	0001	0001	0001
0001	0001	0001	0001	0001	0001	0001	0001

El bloque de memoria destinado a datos no-volátiles no entra en el proceso de descarga. Este bloque esta comprendido entre las direcciones START_FIRST_PAGE_ADDR y END_FIRST_PAGE_ADDR.

10. Calcula la suma en 16-bits de la partición de código de usuario, desde la dirección START_USR_CODE_SECT a END_USR_CODE_SECT. Luego, almacena la suma en complemento a 1, en las direcciones START_CHK_CODE: START_CHK_CODE+1. El byte más significativo se encuentra en la dirección START_CHK_CODE.

3201	e601	4001	e601	4f01	e601	5d01	e601
6601	e601	7601	e701	4901	e701	5301	e701
5a01	0001	0401	0201	0001	0001	0001	0001
0001	0001	0001	0001	0001	0001	0001	0001
0001	0001	0001	0001	0001	0001	0001	0001
0001	0001	0001	0001	0001	0001	0001	0001
0001	0001	0001	0001	0001	0001	0001	0001
0001	0001	0001	0001	0001	0001	0001	0001
0001	0001	0001	0001	0001	0001	0001	0001
0001	0001	0001	0001	0001	0001	0001	0001
0001	0001	0001	0001	0001	0001	0001	0001
0001	0001	0001	0001	0001	0001	0001	0001
0001	0001	0001	0001	0001	0001	0001	00d0

Checksum

11. De acuerdo con el protocolo FC prepara el archivo **hcs08im** con los comandos que procesará el bootloader al descargar una imagen a FLASH del MCU. El archivo **reclog** también registra estos comandos de forma descriptiva, como lo muestra la siguiente figura.

```
erase_blk: 0xFC00

prg_blk: 0xFC00-0xFC3F
blocklen: 0x40 |
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

prg_blk: 0xFC40-0xFC7F
block len: 0x40 |
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

El formato de los comandos del protocolo FC se lista a continuación:

Comandos	Campos		Longitud	Descripcion
WRITE	Command	ASCII 'W'	1	Comando de programación de FLASH
	Data	Address MSB	1	Byte más significativo de la dirección en donde comienza la programación de datos en FLASH
		Address LSB	1	Byte menos significativo de la dirección en donde comienza la programación de datos en FLASH
		Data length	1	Indica la cantidad de datos a programar
		Data	n	Datos a programar
ERASE	Command	ASCII 'E'	1	Comando de borrado de sector de FLASH
	Data	Address MSB	1	Byte más significativo de la dirección perteneciente al sector de FLASH que se desea borrar
		Address LSB	1	Byte menos significativo de la dirección perteneciente al sector de FLASH que se desea borrar
QUIT	Command	ASCII 'Q'	1	Fin de la programación de FLASH

La siguiente figura muestra un fragmento del archivo **hcs08im**. Los datos marcados en color rojo indican un comando del protocolo FC, las dirección de programación y borrado se marcan en color azul, mientras que en color verde la cantidad de datos que transporta el comando WRITE.

Erase command		Erase command address		Write command		Write command address		Data count	
4580	0057	8000	408b	899e	fe05	f6af	019e		
ff05	888a	81a7	fcc6	8085	4c95	e701	c680		
844c	f732	8086	201f	898b	f687	e602	4c9e		
e706	e603	ee01	8a4c	2003	7faf	014b	fb9e		
6b05	f78a	88af	0457	8040	409e	6b02	dd9e		
6b01	d932	8088	898b	adb1	974c	9ee7	03ad		
aa4c	9ee7	044a	2603	5100	18ad	9e87	8aad		
9a97	2005	ad95	f7af	019e	6b04	f79e	6b03		
f320	d5a7	0681	4508	7f94	ad57	8080	408d		
cc80	9200	0180	8ae7	f301	0006	0600	0000		
00cd	92da	cdb3	05cd	938e	cd91	b320	f5a0		
30a1	0923	038c	5f81	ae01	8c81	a020	a15e		

12. Genera el archivo **s08imdwr** compuesto por el archivo **hcs08im** y su suma de verificación en 16-bits complemento a 1. Esta suma reside en las dos primeras posiciones del archivo **s08imdwr**, en donde la primera almacena el byte más significativo. La cadena de versión del archivo se encuentra en la posición **START_FILE_VERSION** cuya longitud es **VERSION_STRING_SIZE**.

Este archivo reside en memoria DataFlash para su posterior lectura desde el bootloader residente.

La siguiente figura muestra un fragmento del archivo **s08imdwr**. Los datos en color rojo que se encuentra a partir de **START_FILE_VERSION**, componen la cadena de versión **"SW00.14.00HW02.00\0"**.

File checksum				START_FILE_VERSION			
e2e1	4518	2c57	182c	1453	5730	302e	3134
2e30	3048	5730	322e	3030	0000	0057	1840
4000	0000	0000	0000	0000	0000	008b	899e
fe05	f6af	019e	ff05	888a	81a7	fcc6	18d1

13. A continuación se procede a abrir el archivo s19 del bootloader. Para generar ahora una imagen completa (aplicación de usuario + bootloader). Esta imagen será utilizada durante el proceso de fabricación, para obtener un equipo funcionando según lo que requiere la aplicación en cuestión.

El archivo s19 del bootloader se procesa registro por registro de manera similar a lo descrito en los puntos anteriores.

14. En este punto, el archivo imagen contiene, entre las direcciones START_REPRO_AREA y END_REPRO_AREA, el código de bootloader más el de aplicación de usuario. Se procede entonces, a barrer nuevamente el contenido de dicho archivo, ahora para generar un archivo s19 con el total de la imagen a cargar, **totalim.s19**.

Port del preprocesador s19

Si se desea cambiar el mapa de memoria debe configurarse el archivo **hc08sprg.h**, con las constantes que se definen en la Tabla-1.

Conversor S19

El conversor de archivos s19 es la herramienta utilizada para realizar las tareas de procesamiento sobre dichos archivos, descritas anteriormente en este documento. Posee la siguiente interfaz de línea de comandos.

Opción	Caracter	Descripción
-c	Obligatorio	Especifica el nombre del archivo s19 de la aplicación de usuario. Genera el archivo s08imdwr .
-b	Opcional	Especifica el nombre del archivo s19 del bootloader. Se genera el archivo totalim.s19 .
Ninguna o solo -b		Muestra la descripción del comando.

Tabla 2. Conversor S19

Si se desea generar unicamente el archivo **s08imdwr** la línea de comandos desde consola es la siguiente: **boot -c usrcode.s19**. Ahora, si se desea generar el archivo **totalim.s19** la línea de comandos desde consola es la siguiente: **boot -c hcs08.s19 -b boot.s19**.

Los archivos de salida generados por el procesador s19 se listan a continuación y se ofrece una breve descripción de los mismos.

Archivo	Descripción
image	Imagen de la memoria FLASH del MCU.
reclog	Especifica los registros del archivo s19. Agrega los comandos del protocolo FC en formato legible.
binlog	Imagen binaria del segmento USR_CODE_SECT.
s08imdwr	Comandos del protocolo FC y el checksum del archivo. Este archivo se utiliza para la reprogramación del segmento USR_CODE_SECT.
hcs08im	Comandos del protocolo FC.

Tabla 3. Archivos de salida