ST10 Flasher DLL

Document Name:

Revision:	**1.01**
Date:	**28-05-01**
Status:	**Draft**
Author/Owner:	**L.Regnier**
Email:	**L.Regnier@st.com**
Organization:	**TPA/DMD/Tools/Emulators**

History:

Version	Date	Author	Comments
1.00		S.Legrix	Creation
1.01	28/05/2001	**L.Regnier**	Release for version 2.2 of ST10 Flasher

T.P.A Division
DMD Group
STMicroelectronics
60, rue Lavoisier – 38330 Montbonnot St. Martin - France

Table of contents

1. Purpose.

This document describes how to handle the **"st10flasher.dll"** file to interface it with an external application. It describes all available functions, which can be useful to design a flasher tool.

2. Products.

The flash products concerned by the dll are: **ST10F167, ST10F168, ST10F169, ST10F269** and **ST10F280**.
External flashes **M29F40 and M29F400** are also supported.
Each product needs a specific monitor file, which is stored in the folder "monitor".
The files "**startidchip.hex**" and "**st10noflash.hex**" needs also to be in the folder "monitor", they are used during the startup for the auto detection of the target.
The auto detection is based on the recognition of the BSL acknowledge and the chip ID. It works only if you want to program the internal flash of the ST10.
For external flash, you have to force the target (Ex. M29F400B).

Warning: The **ST10Flasher.Dll** must be in one of following path:
The application folder.
The system path.
The user path.

Warning: Monitor files should be located in a child folder **'Monitor'** in the same folder as the **ST10Flasher.Dll**.

DEVICE	**Acknowledge value (2.1)**	**Chip ID (2.2)**	**Project name**	**Monitor file (2.3)**
ST10F168	0xD5	0x00A8x	Monitor168	Monitor001
ST10F169	0xD5	0x00A9x	Monitor168	Monitor001
ST10F269	0xD5	0x10dx	Monitor269	Monitor002
ST10F280	0xD5	0x108x	Monitor280	Monitor003
M29F400	0x00	Don't care	Monitor29f400	monitor29f400
M29F40	0x00	Don't care	Monitor29f40	monitor29f40
Other ST10	Don't care	Don't care	MonitorNoFlash	MonitorNoFlash

2.1. BSL acknowledge

Acknowledge is sent by the ST10 just after the reception of the NULL character. As the ST10F168, ST10F169, ST10F269 have the same acknowledge but not the same programming algorithm, it is not possible to base the auto detection on this value.
If the acknowledge is not recognized, the Standard ST10 monitor will be used. This monitor is not able to program any kind of flash (external/internal).

2.2. Chip ID

This value contents the type of CPU (bits 15-4) and the version of the core (bits 3-0).
The version of the core is ignored because it doesn't make difference for programming.
If the chip ID is not recognized, the Standard ST10 monitor will be used. This monitor is not able to program any kind of flash (external/internal).

2.3. Monitor filename

For confidentiality reasons, the device name must not appear in the monitor file name. That's why the monitor168.hex is renamed in monitor001.hex, and monitor269.hex in monitor002.hex.

3. Dll version table.

Visual Basic – st10flasher.exe	**ST10Flasher.dll**	**Monitor version**
V1.5	V1.6	No version number
V1.6	V1.7	No version number
V1.61	V1.7A	No version number
V1.62	V1.7B (not delivered)	No version number
V2.0	V2.0	No version number
V2.2	May 28, 01	7

4. Auto detection of ST10 Frequency

Knowing at witch frequency the ST10 runs is compulsory for two reasons:

- For the F168, the embedded flash algorithm need the period of the Cpu.
- To change the speed of the serial port on the ST10 side.

The principle of the auto detection is based on the measurement of a known signal. To do this, the PC send a null byte that takes 8 bits and the ST10 measures the length of this pulse.

5. Available Baud rates versus ST10 Frequency

To works well, the baud rates deviation between the PC and the ST10 must be lower than 2%
Grayed boxes mean that the frequency is **Ok at sartup.**
Boxes with 'Ok' text means that frequency is available after **download of monitor**.

BR \ F (MHz)	2.5	5	7.5	10	15	20	25
9600	▨ Ok	▨ Ok	▨ Ok	▨ Ok	▨ Ok	▨ Ok	▨ Ok
14400		▨ Ok	▨ Ok	▨ Ok	▨ Ok	▨ Ok	▨ Ok
19200	▨ Ok	▨ Ok	▨ Ok	▨ Ok	▨ Ok	▨ Ok	▨ Ok
38400	▨ Ok	▨ Ok	▨ Ok	▨ Ok	▨ Ok	▨ Ok	▨ Ok
56000						▨ Ok	▨ Ok
57600			▨ Ok		▨ Ok	▨ Ok	Ok
115200			▨ Ok		▨ Ok		
128000							▨ Ok
230400			▨ Ok		▨ Ok		
256000							▨ Ok
460800					▨ Ok		

Figure 1: Available baud rates versus the ST10 frequency. Grayed boxes are Ok for startup.

6. Auto detection of target

The auto detection is based on the recognition of the BSL acknowledge and the chip ID. It works only if you want to program the internal flash of the ST10.
For external flash, you have to force the target (Ex. M29F400B).

6.1. BSL acknowledge

This acknowledge is send by the ST10 just after the reception of the NULL character. It can have the following value

ST10F168	ST10F169	ST10F269	ST10F280	Other ST10
0xD5	0xD5	0xD5	0xD5	Don't care

As the ST10F168, ST10F169, ST10F269 have the same acknowledge but not the same programming algorithm, it is not possible to base the auto detection on this value.
If the acknowledge is not recognized, the Standard ST10 monitor will be used. This monitor is not able to program any kind of flash (external/internal).

6.2. Chip ID

This value contents the type of CPU (bits 15-4) and the version of the core (bits 3-0).
It can have the following value

ST10F168	ST10F169	ST10F269	ST10F280	Other ST10
0x00A8x	0x00A9x	0x10dx	0x1180	Xxxx

The version of the core is ignored because it doesn't make difference for programmation.
If the chip ID is not recognized, the Standard ST10 monitor will be used. This monitor is not able to program any kind of flash (external/internal).

7. Known problems

Actually the software is not able to work with the ST10F167. This is due to the internal BSL that doesn't respect the standard BSL of the ST10: the XRAM isn't enable (XPERCON=0, SYSCON.2=0) and IDCHIP=0. These points seem not documented.

8. Visual Basic interface. Dll functions.

8.1. Function descriptions

.SetCom

Prototype:
unsigned int SetCom(char *PortName, unsigned int ComSpeed)

Parameters:

PortName:	Communication port name (ex. « com1 »).
ComSpeed:	Communication speed. **See table 1 for available frequency**

Description
This function initializes the serial communication at the beginning of the software. The goal is to fix communication parameters in order to be able to load the ST10 monitor.
It returns 1 if the initialization was OK, 0 otherwise.

.CloseCom

Prototype:
unsigned int CloseCom(void)

Parameters:
None.

Description
This function closes the serial communication.
It returns 1 if the operation was OK, 0 otherwise.

.ComIsKline

Prototype:
unsigned int ComIsKline(void)

Parameters:
None.

Description
This function returns **1** if the a k-line communication has been detected; **0** otherwise.

.AdjustFrequency (obsolete)

Prototype:
double AdjustFrequency(double frequency, int *DevError)

Parameters:

frequency:	Communication port name (ex. « com1 »).
DevError:	Communication speed. See table for available frequency

Description
Warning: Obsolete function. Used **AdjustCpuFrequency** instead.
This function performs ST10 frequency detection and can also adjust frequency.
The "DevError" parameter returns 1 if the operation was OK, 0 otherwise.
If the parameter frequency is 0, the function returns the target frequency CPU (in MHz).
Else it sets frequency CPU to frequency parameter value (in MHz).
Knowing at witch frequency the ST10 runs is compulsory for two reasons :

- For the F168, the embedded flash algorithm need the period of the CPU.
- To change the speed of the serial port on the ST10 side.

The principle of the auto detection is based on the measurement of a known signal. To do this, the PC sends a null byte.

Return value

0 if the function failed. Resetting the target and reloading the monitor is greatly recommended.
1 otherwise.

.AdjustCpuFrequency

Prototype:

double AdjustFrequency(double frequency)

Parameters:

frequency:	New ST10 frequency to set.

Description

The function return true if the deviation between the **frequency** and the internal **CPU frequency** (calculated at startup) is lower than 1%. In that case, the function changes the **CPU frequency** that is used to change the communication speed.

If deviation is >1% the function return false and the **CPU frequency** stays unchanged, because we consider that a wrong input has been done and a bad can crash the serial communication.

Return value

0 if the function failed
1 otherwise.

.SetComSpeed

Prototype:

unsigned int SetComSpeed(char *PortName, unsigned int ComSpeed)

Parameters:

PortName:	Communication port name (ex. « com1 »).
ComSpeed:	Communication speed. See table for available frequency

Description

Change the serial communication speed between PC and the ST10 Target baud rate. This means that both, the PC and the ST10 board should accept the new baud rate. If the two targets accept the speed change, a test is done to verify that the communication works properly.
If not, the rate is restored to 9600 baux and the two targets try to resynchronize together.

Return value

0 if the function failed. Resetting the target and reloading the monitor is greatly recommended.
1 otherwise.

.TestCom (obsolete)

Prototype:

unsigned int TestCom(void)

Parameters:

None

Description

Tests if the serial communication speed between PC and the ST10 Target works fine. The PC sends a NULL byte and receives an acknowledge from the ST10. This function can be used to test the serial communication and also the monitor

Warning: This function is now obsolete. The has been integrated in the SetComSpeed function.

Return value

0 if the function failed. In that case, it means that the communication is broken between the two parts or the ST10 monitor has crashed. In both cases, resetting the target and reloading the monitor is greatly recommended.
1 Test OK.

.LoadFile

Prototype:

unsigned int LoadFile(char *filename)

Parameters:

Filename: Name of the file to be loaded.

Description

Load the file into the PC memory.

Return value

1 if the loading was OK,
0 otherwise.

.InitMonitor

Prototype:

Prototype : unsigned int InitMonitor(char *target)

Parameters:

target: Address of a string that will contain the name of target (ex: ST10F168) after the return of the function.

Warning: The size of this string must be at least 32 bytes.

Description

This function loads the monitor and initializes the flash parameters. The first action of this function is to make the autodetection of the target. If this action succeeds, the right monitor file will be load and target .will contain the name of the target.

Warning: Monitor files are search under the folder **"Monitor\" in the ST10Flasher.Dll directory.**

Return value

1 if the initialization was OK.
0 otherwise.

.GetST10FlasherVersion

Prototype:

unsigned int GetST10FlasherVersion(char *cBlock)

Parameters:

cBlock: Sting that will receive the version of the flasher.

Warning: This string must contain at least 256 bytes.

Description

Return the internal version of the ST10Flasher DLL.

Return value

1 if OK.
0 otherwise.

.IsAvailableBaudRate

Prototype:

unsigned int IsAvailableBaudRate(const double frequency, unsigned int baudrate)

Parameters:

frequency: ST10 cpu frequency.
Baudrate: Serial baudrate that you want to test.

Description

Return 1 the deviation for BaudRate is low enough to allow a good reliability of the communication between the PC and the ST10.

Return value

1 if OK.
0 otherwise.

.IsMonitorAlive

Prototype:

unsigned int IsMonitorAlive(void)

Parameters:

None

Description

Return 1 if the communication works well between the PC, ST10 and monitor.
If the function return 0 it means that something wrong has append and you have to reset the ST10 and reload the monitor.

Return value

1 if OK.
0 otherwise.

.AutoROMS1

Prototype:

unsigned int AutoROMS1(unsigned int &SegNumber)

Parameters:

SegNumber: If the function succeeds, **SegNumber** will contain the polarity of **Syscon.ROMS1** bit.

Description

Try to see if Syscon.ROMS1 can be set automatically or not. This bit can be set if there isn't simultaneously code in segment 0 and 1. In that case SegNumber will contain the right value.
If the code contains simultaneously datas in segment 0 and 1, the function **SetRomS1** has to be called to force **Syscon.ROMS1** to the desired value.

Return value

1 if the autodetection was OK.
0 otherwise.

.SetROMS1

Prototype:

void SetROMS1(unsigned int ROMS1).

Parameters:

If **ROMS1 =0**, the first bank of the flash will be located at **address 00000**.
If **ROMS1= 1**. the first bank of the flash will be located at **address 0x10000**.

Description

This function sets or clears the **Syscon.ROMS1** bit of the ST10.
This function should only to be call when codes are present simultaneously in segment 0 and 1. In that case the AutoRomS1 function cannot finds which polarity to be given to **Syscon.ROMS1**. So in that case, call SetRomS1 to force this bit.

Return value

None

.EraseFlash

Prototype:

unsigned int EraseFlash(unsigned int BlockMask)

Parameters:

BlockMask: Block is in hexadecimal format (i.e.: to erase blocks 0 and 1, BlockMask will be 3).

Description

This function erases the memory area defined by the value **BlockMask**.

Return value

1 if the Erase was OK.
0 otherwise.

.VerifyFlash

Prototype:

unsigned int VerifyFlash(unsigned int BlockMask)

Parameters:

BlockMask: BlockMask is in hexadecimal format (i.e.: to verify blocks 0 and 2, Block will be 5).

Description

This function checks if the memory area defined by the value BlockMask corresponds to the current file loaded.

Return value

1 if the initialization was OK.
0 otherwise.

.BlockNbToErase

Prototype:

unsigned int BlockNbToErase(bool ErrorToSet)

Parameters:

ErrorToSet: If non zero, an error message occurs when there is block(s) to erase. Else no error message will be set.

Description

Compute the intersection between the non-blank blocks and the blocks concerned by the current file loaded.

Return value
This function returns blocks that need to be erased (9 means that blocks 0 and 3 have to be erased).

.GetHexFileBlock

Prototype:
unsigned int GetHexFileBlock(unsigned int &BlockMask)

Parameters:
BlockMask: If the function succeeds, BlockMask will contain blocks concerned by the hexfile.

Description
Computes which blocks are concerned in the hexfile currently loaded in memory.

Return value
1 if OK.
0 otherwise.

.ProgramAndVerify (obsolete)

Prototype:
unsigned int ProgramAndVerify(void)

Parameters:
None.

Description
program the flash and verify the programming.
Warning: This function is obsolete. Used ProgramFlash instead

Return value
1 if the programmation and the verification was OK.
0 otherwise.

.ProgramFlash

Prototype:
Prototype : unsigned int ProgramFlash(void)

Parameters:
None.

Description
This function programs and verify the flash with the current loaded file.

Return value
1 if the programmation and the verification was OK.
0 otherwise.

.GetError

Prototype:
unsigned int GetError(char *BufferForStatus)

Parameters:
BufferForStatus. String to get back the description of the error.
Warning: The size of **BufferForStatus** must be at least 256 bytes.

Description

When errors are detected, **GetError** returns non zero value and **BufferForStatus** contains the first error that has occurred
All pending errors are cleared.

Return value
1 if an error has occured.
0 if no error has occured.

.BlankCheck (obsolete)

Prototype:
unsigned int GetError(char *BufferForStatus)

Parameters:

Description

Return value
1 if OK.
0 otherwise.

Prototype:
unsigned int BlankCheck(void)

Parameters:
None

Description
This function is returns a value indicating if the different blocks are blank or not.
If bit number i of the returned value is set it means that block i is blank, else block i is not blank.
Warning: It's better to use **GetBlankBank.**

Return value
If an error occurs, a code error (value = 240) is returned.
Else returns non blank blocks mask (6 means that blocks 1 and 2 are blanked).

.GetBlankBank

Prototype:
unsigned int GetBlankBank(unsigned int &BlanckBlockMask))

Parameters:
BlanckBlockMask: If the function succeeds, **BlanckBlockMask** will contain blank blocks (6 means that blocks 1 and 2 are blanked)..

Description
Gives blank blocks mask. It's the same function as BlankCheck, only prototype has changed.

Return value
1 if OK.
0 otherwise.

.EnablePort

Prototype:
unsigned int EnablePort(UWordT PortNb, UWordT BitPosition,bool Val)

Parameters:
PortNb: The port number.
BitPosition: The bit number of the port

Val: 1 to set the port, 0 to reset it.

Description

Put **PortNb.BitPosition** pin in output state with the **Val** polarity (0 or 1).

Return value

1 if OK.
0 otherwise.

.DisablePort

Prototype:

unsigned int DisablePort(UWordT PortNb, UWordT BitPosition)

Parameters:

PortNb: The port number.
BitPosition: The bit number of the port

Description

This function is devoted to disable the port PortNb,BitPosition. To disable it, the port is configured as input.

Return value

1 if OK.
0 otherwise.

.DumpBlock (obsolete)

Prototype:

unsigned int DumpBlock(char *filename, unsigned int BlockNb, unsigned int format)

Parameters:

Filename: Name of the output file.
BlockNb: Mask for blocks to dump.
Format: output format: 1 for byte, 2 for Word, 4 for Long word.

Description

This function dumps the flash memory area defined by BlockNb in a file *filename with the specified format. Dump is done in ASCII format
If the bit i of the parameter BlockNb is set, block i will be dumped.

Warning: This function is obsolete. Used **DumpBlocks** instead

Return value

1 if OK.
0 otherwise.

.DumpOffset (obsolete)

Prototype:

unsigned int DumpOffset(char *filename, unsigned int BlockNb, unsigned int format, unsigned int Offset)

Parameters:

Filename: Name of the output file.
BlockNb: Mask for blocks to dump.
Format: output format: 1 for byte, 2 for Word, 4 for Long word.
Offset: Offset of the start address

Description

Dump in ASCII format the content of the flash memory. The size is limited to 4096 elements.

Warning: This function is obsolete. Used **DumpBlocks** instead

Return value

1 if OK.
0 otherwise.

.DumpAddress (obsolete)

Prototype:

DumpAddress(char *filename, unsigned int StartAddr, unsigned int Size, unsigned int format)

Parameters:

Filename: Name of the output file.
StartAddr: StartAddress for the dump.
Size: Number of elements to dump (the size of 'element' depends of **Format**).
Format: output format: 1 for byte, 2 for Word, 4 for Long word.

Description

Warning: This function is obsolete. Used **DumpBlocks** instead

Return value

1 if OK.
0 otherwise.

.DumpBlocks

Prototype:

unsigned int DumpBlocks(char *filename, unsigned int BlockNb, unsigned int Option)

Parameters:

Filename: Output filename (if still exists, it will be overwritten).
BlockNb: Block mask for the dump.
Option: Description of the output format
    Bits 3-0: Size of element 0=Byte, 1= Word, 2= LongWord…
    Bits 7-4: 0=IntelHex, 2=Text, other values not supported.

Description

Dump the content of the flash in a file by block.

Return value

1 if OK.
0 otherwise.

.DumpRange

Prototype:

unsigned int DumpRange(char *filename, ULWordT Start, ULWordT Size, ULWordT Option)

Parameters:

Filename: Output filename (if still exists, it will be overwritten).
Start: StartAddress for the dump.
Size: Number of elements to dump (the size of 'element' depends of **Option.Format**).
Option: Description of the output format
    Bits 3-0: Size of element 0=Byte, 1= Word, 2= LongWord…
    Bits 7-4: 0=IntelHex, 2=Text, other values not supported.

Description

Dump the content of the flash in a file by address.

Return value

1 if OK.
0 otherwise.

.GetNbBlock

Prototype:

unsigned int GetNbBlock(void)

Parameters:

None.

Description

Return the number of blocks in the current flash.

Return value

the number of blocks in the current flash.

.GetBlockDescription

Prototype:

unsigned int GetBlockDescription(const unsigned int BlockNb, char *cBlock)

Parameters:

BlockNb: Block number.
cBlock : String .that receives the description of the block.
Warning: This string must contain at least 256 bytes.

Description

If **BlockNb** is a valid block number, the function returns in the string **cBlock** the description of this block: name, size and range(ex cBlock='Bank1: 16k (0000-3FFF').

Return value

1 if OK.
0 otherwise.

.GetBlockMaskDescription

Prototype:

unsigned int GetBlockMaskDescription(const unsigned int BlockMask, char *cBlock, const char *Header)

Parameters:

BlockMask: Mask for blocks.
cBlock : String .that receives the description of the block.
Warning: This string must contain at least 256 bytes.
Header:'Header' for each block (if BlockMask=5 and Header='bank', the result will be 'bank0, bank2').

Description

Convert a block mask to a string. This function can be used for example after a **GetBlankBank** called to display which bank are erased or not.

Return value

1 if OK.
0 otherwise.

.GetBlockRange

Prototype:

unsigned int GetBlockRange (const unsigned int BlockNb , char * Description, unsigned int& Start , unsigned int& Size, unsigned int First)

Parameters:

BlockNb: Block number to retrieve information.
Description: String that will received the description of the block.
Warning: This string must contain at least 256 bytes.
Start: Will receive the start address of the block.
Size: Will receive the size of the block
First:This parameter is used when a block can be split in two ranges (for example the block1 of the ST10F168 is 4000-7FFF, 18000-1FFFF). The first call to GetBlockRange with **First**=1 will return 4000-7FFF and a second call to this function with **First=false and Start and Size that still contain previous** results will return 18000-1FFFF. In fact, when **First**=0, the function return the first available address for the block after (Start+Size).
By default First must be set to 1.

Description

If BlockNb is valid, return the description, start address and size of the desired block.

Return value

1 if OK (valid block number).
0 otherwise.

.Alignement

Prototype:

unsigned int Alignement(unsigned int&Addr, unsigned int Align, unsigned int ToUpper)

Parameters:

Addr: Address to be rounded. If the function succeeds, this address will be modified.
Align: size of the alignement. 0=Byte, 1= Word, 2= Double word…..
ToUpper: If 1 rounds to upper value.

Description

Round an address to upper or lower bound regarding the Align parameter.
Ex: if align=2 (double word) and address 0x1FF6
ToUpper=0 → Address=0x1FF4.
ToUpper=1 → Address=0x1FF7.

Return value

1 if OK (valid block number).
0 otherwise.

8.2. New functions

Following functions have been added in the release 2.2 of the DLL. They are not compulsory to program ST10 flashs but it can help developers.

- unsigned int GetNbBlock(void)
- unsigned int GetBlockDescription(const unsigned int BlockNb, char *cBlock)
- unsigned int GetBlockMaskDescription(const unsigned int BlockMask, char *cBlock, const char *Header)
- unsigned int GetST10FlasherVersion(char *cBlock)

- unsigned int GetBlankBank(unsigned int &BlanckBlockMask)
- unsigned int CloseCom(void)
- unsigned int IsAvailableBaudRate(const double frequency, unsigned int baudrate)
- unsigned int IsMonitorAlive(void)
- unsigned int DumpBlocks(char *filename, unsigned int BlockNb, unsigned int Option)
- unsigned int DumpRange(char *filename, ULWordT Start, ULWordT Size, ULWordT Option)
- unsigned int GetBlockRange (const unsigned int BlockNb , char * Description, unsigned int& Start , unsigned int& Size, unsigned int First)
- unsigned int Alignement(unsigned int& Addr,unsigned int Align,unsigned int ToUpper)

8.3. obsolete functions

Following functions are now obsolete and won't be supported later. Try to replace them by their equivalent if possible.

- unsigned int BlankCheck(void) // --> Replaced by GetBlankBank
- unsigned int Close(void) // --> Replaced by CloseCom
- unsigned int ProgramAndVerify(void)// --> Replaced by ProgramFlash.
- double AdjustFrequency(double frequency, int *DevError)--> Replaced by AdjustCpuFrequency
- unsigned int TestCom(void)
- unsigned int TestComSpeed(unsigned int *i,unsigned int *WrongData)
- unsigned int DumpToScreenAddress(char *filename, unsigned int StartAddr, unsigned int Size, unsigned int format) --> Replaced by DumpBlocks
- unsigned int DumpAddress(char *filename, unsigned int StartAddr, unsigned int Size)--> Replaced by DumpBlocks
- unsigned int DumpOffset(char *filename, unsigned int BlockNb,unsigned int format, unsigned int Offset) --> Replaced by DumpBlocks