AN4429
Application note

RPC56xx and SPC56xx C90FL Flash recovery in case of brownout during Flash erase operation

Introduction

The RPC56xx and SPC56xx families of devices have internal Flash used for code and data. The following RPC56xx and SPC56xx devices use C90FL technology they are for internal Flash:

- RPC56ELxx
- RPC564Axx
- SPC56xLxx
- SPC564Axx

If, on C90FL Flash, there is an accidental power loss or supply voltage drop or unexpected reset (in general called brownout) that happens during a Flash erase operation, the Flash blocks erased are left a not deterministic state.

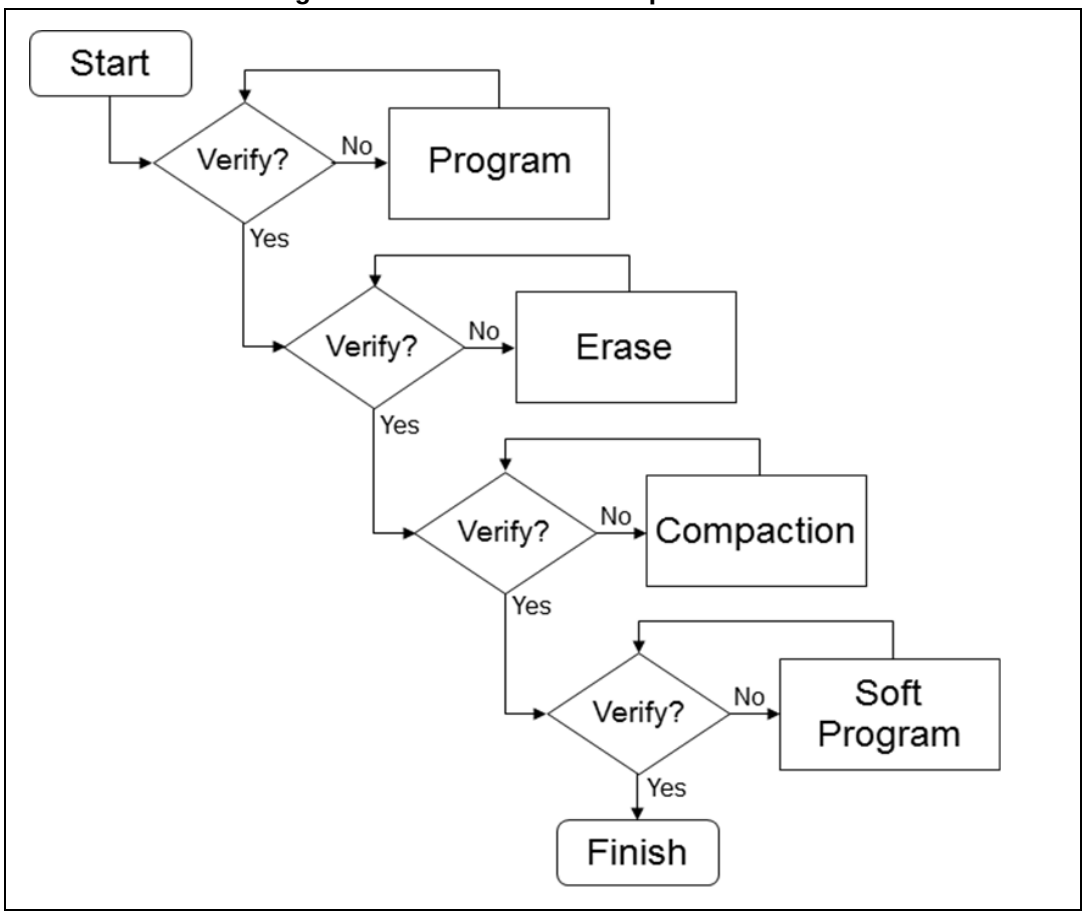
This application note describes how to recover the C90FL Flash block(s) interrupted during an erase operation in case of such brownouts.

Contents

1 C90FL Flash erase operation

C90FL Flash erase operation consists of multiple steps as shown in *Figure 1* which is implemented by the C90FL Flash memory controller hardware. First, all bits in the selected Flash block(s) are programmed in order to verify the level that allows the erase function to start at consistent state. Then the erase step deletes all bits. Since the erase is a bulk operation in which an erase pulse moves all the bits in a Flash block, some bits are over-erased, for example, under the compaction of check up or under a software program of verify level. At the compaction step the over-erased columns are compacted back to reduce column leakage. And finally all the bits below software program verify level are reprogrammed with very low gate, in order to avoid overshoot of erase verify level for any bits. As a result, when an erase operation is completed, all the bits in the selected blocks have their threshold voltage within a pre-defined window, between erase verify level and software program verify level.

Figure 1. C90FL Flash erase operation flow



2 Issue caused by brownout during Flash erase

If there is an accidental power loss or supply voltage drop or unexpected reset on C90FL Flash, (i.e., brownout) that occurs during a Flash erase operation, the Flash blocks that are erased are left a not deterministic state, but it depends on the step where the erase operation has been interrupted.

Typically to recover the Flash block(s) that are interrupted to a working state again, users can simply perform an erase operation on the Flash block(s). However, there are two cases that require special attention.

2.1 Recover from non-correctable ECC errors

If a brownout occurs during Flash erase operation the bits in the block(s) remain and non-correctable ECC (error correcting code) errors appear. Note that in C90FL Flash, a single-bit error correction and double-bit error detection (SEC-DED) ECC code are used.

For example, if the brownout occurs during the program step in the erase operation, many Flash pages, including the corresponding ECC bits, are left programmed. Note that all zeros are not a valid ECC codeword, and hence this causes non-correctable ECC errors when they are reading those Flash block(s). Similarly the brownout can occur after the program step but in the middle of the erase step.

Even at this state, the Flash block(s) can still be erased to recover it. However, some Flash programmer tools (e.g. Lauterbach) may read Flash while executing a code from RAM before performing an erase operation. As a result, it generates ECC error and hence an exception, and if the Flash programmer tool does not have proper exception handler implemented, it may cause the code execution to hang in the Flash programmer tool and thus cause the erase operation to fail.

So to recover from this state, users need to be aware of possible Flash programmer tool failure for erase operation caused by Flash ECC exception. Users can either use the FlashErase function provided in the RPC56xx/SPC56xx C90FL Flash Standard Software Driver to erase the interrupted Flash block(s), or use Nexus/JTAG debugger script to simply toggle the low-level Flash register bits to perform the erase operation. Please refer to *Appendix A* for an example Lauterbach Trace32 script for erasing C90FL Flash.

2.2 Recover from depleted bits

It is also possible that a brownout during Flash erase operation leave the bits in the Flash block(s) erased at an over-erased or depleted state. For instance, if the brownout occurs after the erase step but before the compaction and software program step to complete, it is possible to be left in that state.

Depending on how much depleted the bits are, the excessive column leakage caused by the bits might cause the following program operation to fail due to suppressed drain bias. Note that the first step in an erase operation is a program step, and thus for this case, the erase operation to recover the interrupted Flash block(s) might fail. This appears to users as if the Flash block(s) cannot be erased and recovered.

To recover the depleted Flash block(s) for this case, users need to use the binary C-array FlashDepletionRecover function provided in RPC56xx/SPC56xx C90FL Flash Standard Software Driver to recover the depleted bits in the Flash block(s) first, and then re-start an erase operation to recover the block(s). The FlashDepletionRecover is called in the similar way as any other binary C-array Flash driver function. Please refer to *Appendix B* for an example code of calling this function to recover the depleted blocks.

3 Conclusion

The Flash erase operation can be used to recover the Flash block(s) being interrupted due to brownout during erase operation. However, it is considered for cases with non-correctable ECC error and bits left at a depleted state. A new FlashDepletionRecover function has been developed to enable users to recover Flash block(s) left in a depleted state due to brownout.

Appendix A Example script

Example Lauterbach Trace32 script for erasing C90FL Flash on SPC564Lxx:

```
;lblk and hblk are inputs (corresponding to LMS and HBS registers)
to select which block(s) to be erased

local &mcr &peg &lblk &hblk &mcr &fdone

entry &lblk &hblk


;reset MCR

d.s ea:0xc3f88000 %long 0

d.s ea:0xc3f88000 %long 0


;enable block

d.s ea:0xc3f88010 %long &lblk

d.s ea:0xc3f88014 %long &hblk

&fdone=0


;set ERS

d.s ea:0xc3f88000 %long 0x4


;interlock write

d.s ea:0x00000000 %long 0xffffffff


;set EHV

d.s ea:0xc3f88000 %long 0x5


;wait for DONE

while &fdone==0

(

&mcr=D.L(ea:0xc3f88000)

&fdone=&mcr&0x0400

)


;clear EHV

d.s ea:0xc3f88000 %long 0x4
```

```
&mcr=D.L(ea:0xc3f88000)
&peg=&mcr&0x0200

if &peg==0
print "fail"
else
print "pass"

;clear ERS
d.s ea:0xc3f88000 %long 0
```

Appendix B Example code

Example code of calling the FlashDepletionRecover function to recover depleted blocks.

```
#include    "ssd_types.h"
#include    "ssd_c90fl.h"
#include    "normaldemo.h"

extern const unsigned int FlashInit_C[];
extern const unsigned int FlashDepletionRecover_C[];
extern const unsigned int SetLock_C[];

/* Assign function pointers */
pFLASHINIT     pFlashInit     = (pFLASHINIT)     FlashInit_C;
pSETLOCK       pSetLock       = (pSETLOCK)       SetLock_C;
pFLASHDEPLETIONRECOVER    pFlashDepletionRecover    =
(pFLASHDEPLETIONRECOVER)    FlashDepletionRecover_C;

SSD_CONFIG ssdConfig = {
    C90FL_REG_BASE,          /* c90fl control register base */
    MAIN_ARRAY_BASE,         /* base of main array */
    0,                       /* size of main array */
    SHADOW_ROW_BASE,         /* base of shadow row */
    SHADOW_ROW_SIZE,         /* size of shadow row */
    0,                       /* block number in low address space */
    0,                     /* block number in middle address space */
    0,                       /* block number in high address space */
    0x10,                    /* page size */
    FALSE                    /* debug mode selection */
};

UINT32 main(void)
{
    UINT32 returnCode;           /* Return code from each SSD
function. */
```

```
    BOOL   shadowFlag;           /* shadow select flag */
    UINT32 lowEnabledBlocks;     /* selected blocks in low space */
    UINT32 midEnabledBlocks;     /* selected blocks in middle space
*/
    UINT32 highEnabledBlocks;    /* selected blocks in high space */
/*========================= Initialize Part==================*/
returnCode = pFlashInit( &ssdConfig );
    if ( C90FL_OK != returnCode )
    {
        ErrorTrap(returnCode);
    }

    /* Unlock all main array blocks */
    returnCode = pSetLock( &ssdConfig, LOCK_LOW_PRIMARY, 0,
FLASH_LMLR_PASSWORD);
    if ( C90FL_OK != returnCode )
    {
        ErrorTrap(returnCode);
    }
    returnCode = pSetLock( &ssdConfig, LOCK_LOW_SECONDARY, 0,
FLASH_SLMLR_PASSWORD);
    if ( C90FL_OK != returnCode )
    {
        ErrorTrap(returnCode);
    }
    returnCode = pSetLock( &ssdConfig, LOCK_MID_PRIMARY, 0,
FLASH_LMLR_PASSWORD);
    if ( C90FL_OK != returnCode )
    {
        ErrorTrap(returnCode);
    }
    returnCode = pSetLock( &ssdConfig, LOCK_MID_SECONDARY, 0,
FLASH_SLMLR_PASSWORD);
    if ( C90FL_OK != returnCode )
    {
        ErrorTrap(returnCode);
    }
```

```
    returnCode = pSetLock( &ssdConfig, LOCK_HIGH, 0,
FLASH_HLR_PASSWORD);
    if ( C90FL_OK != returnCode )
    {
        ErrorTrap(returnCode);
    }
/*=============== Deletion recover main array space =========*/
shadowFlag = FALSE;
    /* Select the all main array blocks */
    lowEnabledBlocks = 0xffffffff;
    midEnabledBlocks = 0xffffffff;
    highEnabledBlocks = 0xffffffff;

    returnCode = pFlashDepletionRecover( &ssdConfig, shadowFlag,
lowEnabledBlocks, midEnabledBlocks, highEnabledBlocks,
NULL_CALLBACK );
    if ( C90FL_OK != returnCode )
    {
        ErrorTrap(returnCode);
    }
/*=================== DEMO FINISHED ======================*/
/* DEMO PASSED */
    return ((UINT32)DEMO_PASS);
}

/*****************************************************************
| function implementations (scope: module-local)
|----------------------------------------------------------------*/
/* Error trap function */
void ErrorTrap(UINT32 returnCode)
{
    VUINT32 failedReason;

    failedReason = returnCode;

    while(1)
```

```
    {
        ;
    }
}
```

The function pointers for the driver functions are defined in “ssd_c90fl.h” as below:

```
typedef UINT32 (*pFLASHINIT) ( PSSD_CONFIG pSSDConfig );

typedef UINT32 (*pFLASHDEPLETIONRECOVER) (
                PSSD_CONFIG pSSDConfig,
                BOOL shadowFlag,
                UINT32 lowEnabledBlocks,
                UINT32 midEnabledBlocks,
                UINT32 highEnabledBlocks,
                void (*CallBack)(void)
               );

typedef UINT32 (*pSETLOCK) (
                PSSD_CONFIG pSSDConfig,
                UINT8 blkLockIndicator,
                UINT32 blkLockState,
                UINT32 password
               );
```

Revision history

Table 1. Document revision history

Date	Revision	Changes
13-Jan-2014	1	Initial release.
23-Jan-2015	2	Added RPC56xx RPNs and new Disclaimer.