

# Lecture et écriture dans la mémoire programme d'un PIC32MX795F512L

Ce complément présente la gestion de la lecture et de l'écriture dans la mémoire programme (mémoire flash) d'un PIC32MX795F512L.

Ce document a été établi sur la base de Harmony v1.06.

## AIDE HARMONY (NVM PERIPHERAL LIBRARY)

Au niveau de l'aide de Harmony, c'est la section NVM Peripheral Library qui sert de référence. En plus de l'aide, nous disposons d'un exemple sous C:\microchip\harmony\v<n>\apps\examples\peripheral\flash\flash\_modify

## FLASH MEMORY FUNCTIONS

On dispose des fonctions suivantes pour l'accès à la mémoire flash :

|   | Name  | Description   |
|---|---|---|
| → | <a href="#">PLIB_NVM_DataBlockSourceAddress</a>           | Takes the address parameter in the argument and loads the base address from which data has to be copied into flash. |
| → | <a href="#">PLIB_NVM_FlashAccessEnable</a>                | Allows access to the Flash program memory   |
| → | <a href="#">PLIB_NVM_FlashAddressToModify</a>             | Modify the address  |
| → | <a href="#">PLIB_NVM_FlashEraseOperationSelect</a>        | Performs erase operation on the memory row selected   |
| → | <a href="#">PLIB_NVM_FlashEraseStart</a>                  | Performs erase operation on the selected memory area  |
| → | <a href="#">PLIB_NVM_FlashProvideData</a>                 | Provides the data to be written into flash  |
| → | <a href="#">PLIB_NVM_FlashProvideQuadData</a>             | Provides the quad data to be written into flash   |
| → | <a href="#">PLIB_NVM_FlashRead</a>                        | Read the specified address of Flash.  |
| → | <a href="#">PLIB_NVM_FlashWriteCycleHasCompleted</a>      | This routine provides the status of the Flash/EEPROM write cycle.   |
| → | <a href="#">PLIB_NVM_FlashWriteKeySequence</a>            | Copies the mandatory KEY sequence into the respective registers.  |
| → | <a href="#">PLIB_NVM_FlashWriteOperationSelect</a>        | Performs erase operation on the memory row selected   |
| → | <a href="#">PLIB_NVM_FlashWriteProtectMemoryAreaRange</a> | Set the address below which physical memory will be write protected   |
| → | <a href="#">PLIB_NVM_FlashWriteStart</a>                  | Performs erase operation on the memory row selected   |
| → | <a href="#">PLIB_NVM_IsProgramFlashMemoryLocked</a>       | Provides lock status of Program Flash write protect register.   |
| → | <a href="#">PLIB_NVM_ProgramFlashBank1LowerRegion</a>     | Maps the bank 1 to lower mapped region  |
| → | <a href="#">PLIB_NVM_ProgramFlashBank2LowerRegion</a>     | Maps the bank 2 to lower mapped region  |

On dispose des fonctions suivantes pour la configuration :

|   | Name   | Description   |
|---|--|---|
| → | <a href="#">PLIB_NVM_LowVoltageEventsActive</a>      | Provides low voltage detection status   |
| → | <a href="#">PLIB_NVM_LowVoltageIsDetected</a>        | Provides low voltage error detection status                                       |
| → | <a href="#">PLIB_NVM_MemoryModifyEnable</a>          | Allows write cycles to Flash/EEPROM   |
| → | <a href="#">PLIB_NVM_MemoryModifyInhibit</a>         | Inhibits write cycles to Flash/EEPROM   |
| → | <a href="#">PLIB_NVM_MemoryOperationSelect</a>       | Selects the operation to be performed on Flash/EEPROM memory.                     |
| → | <a href="#">PLIB_NVM_StopInIdleDisable</a>           | Continues Flash operation when device enters idle mode.                           |
| → | <a href="#">PLIB_NVM_StopInIdleEnable</a>            | Discontinues Flash operation when device enters idle mode.                        |
| → | <a href="#">PLIB_NVM_WriteOperationHasTerminated</a> | This routine provides the status of the Flash/EEPROM write operation or sequence. |

## PRINCIPES D'UTILISATION DES FONCTIONS

L'effacement et l'écriture ne sont pas réalisés par une fonction unique. Il est nécessaire d'utiliser des fonctions pour configurer l'action et une autre fonction pour la déclencher. La flash est gérée à partir d'un groupe de registres agissant comme un interface.

- **NVMCON: Programming Control Register<sup>(1,2,3)</sup>**
- **NVMKEY: Programming Unlock Register**
- **NVMADDR: Flash Address Register<sup>(1,2,3)</sup>**
- **NVMDATA: Flash Program Data Register**
- **NVMSRCADDR: Source Data Address Register**

Par exemple avec la fonction `PLIB_NVM_FlashWriteStart` :

```
void PLIB_NVM_FlashWriteStart(NVM_MODULE_ID index);
```

Il est nécessaire d'établir les éléments suivants :

### Preconditions

- The Address of the page to be written must be provided using `PLIB_NVM_FlashAddressToModify()`.
- Erase Operation should be selected using the API `PLIB_NVM_MemoryOperationSelect`
- The module should be configured to access Flash memory using `PLIB_NVM_MemoryModifyEnable()`.
- Unlock key sequence should be provided using API `PLIB_NVM_FlashWriteKeySequence`.

### LA FONCTION `PLIB_NVM_FLASHADDRESS TOMODIFY`

La fonction `PLIB_NVM_FlashAdressToModify` permet de spécifier à quelle adresse dans la mémoire flash s'exécutera l'action d'écriture ou d'effacement.

```
void PLIB_NVM_FlashAddressToModify(NVM_MODULE_ID index, uint32_t address);
```

Exemple :

```
PLIB_NVM_FlashAddressToModify(NVM_ID_0, NVM_PROGRAM_PAGE);
```

Le paramètre `NVM_ID_0` est à fournir à toutes les fonctions de gestion de la NVM.

👉 L'obtention de l'adresse sera étudiée plus loin dans ce complément.

## LA FONCTION **PLIB\_NVM\_MEMORYOPERATIONSELECT**

La fonction `PLIB_NVM_MemoryOperationSelect` permet de spécifier l'opération à effectuer dans la mémoire flash.

```
void PLIB_NVM_MemoryOperationSelect(NVM_MODULE_ID index,  
                                     NVM_OPERATION_MODE operationmode)
```

Le type énuméré `NVM_OPERATION_MODE` liste les opérations possibles.

```
typedef enum {  
    WORD_PROGRAM_OPERATION = 0x1,  
    ROW_PROGRAM_OPERATION = 0x3,  
    PAGE_ERASE_OPERATION = 0x4,  
    FLASH_ERASE_OPERATION = 0x5,  
    NO_OPERATION = 0x0  
} NVM_OPERATION_MODE;
```

On en déduit que pour le PIC32MX, il est possible d'effacer l'entier de la mémoire flash ou une page. Au niveau de l'écriture (programmation), il est possible d'écrire un *row*, ce qui correspond à une découpe de la page en bloc. Il est aussi possible d'écrire par mots de 32 bits.

Exemple :

```
PLIB_NVM_MemoryOperationSelect(NVM_ID_0,  
                                ROW_PROGRAM_OPERATION);
```

## LA FONCTION **PLIB\_NVM\_MEMORYMODIFYENABLE**

La fonction `PLIB_NVM_MemoryModifyEnable` permet d'autoriser la modification du contenu de la mémoire flash.

```
void PLIB_NVM_MemoryModifyEnable(NVM_MODULE_ID index)
```

## LA FONCTION **PLIB\_NVM\_FLASHWRITEKEYSEQUENCE**

La fonction `PLIB_NVM_FlashWriteKeySequence` permet d'écrire la séquence de déverrouillage de la flash.

```
void PLIB_NVM_FlashWriteKeySequence(NVM_MODULE_ID index,  
                                     uint32_t keysequence)
```

### EXEMPLE DE DÉVERROUILLAGE

```
// Write the unlock key sequence  
PLIB_NVM_FlashWriteKeySequence(NVM_ID_0, 0x0);  
PLIB_NVM_FlashWriteKeySequence(NVM_ID_0, 0xAA996655);  
PLIB_NVM_FlashWriteKeySequence(NVM_ID_0, 0x556699AA);
```

Les valeurs à fournir sont tirées de l'exemple fourni par Microchip pour le PIC32MX.

## EXEMPLE D'EFFACEMENT D'UNE PAGE

La fonction NVMPageErase, reprise de l'exemple Microchip et fournie dans les fichiers Mc32NVMUtil.h et Mc32NVMUtil.c, montre une séquence complète d'effacement d'une page.

```
/*
    Function:
        void NVMPageErase (uint32_t address)

    Summary:
        Erases a page in flash memory (4 KB)
*/
void NVMPageErase(uint32_t address)
{
    // Base address of page to be erased
    PLIB_NVM_FlashAddressToModify(NVM_ID_0,
                                   virtualToPhysical(address));

    // Disable flash write/erase operations
    PLIB_NVM_MemoryModifyInhibit(NVM_ID_0);

    // Select page erase function & enable
    // flash write/erase operations
    PLIB_NVM_MemoryOperationSelect(NVM_ID_0,
                                    PAGE_ERASE_OPERATION);

    // Allow memory modifications
    PLIB_NVM_MemoryModifyEnable(NVM_ID_0);

    // Write the unlock key sequence
    PLIB_NVM_FlashWriteKeySequence(NVM_ID_0, 0x0);
    PLIB_NVM_FlashWriteKeySequence(NVM_ID_0, 0xAA996655);
    PLIB_NVM_FlashWriteKeySequence(NVM_ID_0, 0x556699AA);

    // Start the operation
    PLIB_NVM_FlashEraseStart(NVM_ID_0);

    // Wait until the operation has completed
    while (!PLIB_NVM_FlashWriteCycleHasCompleted(NVM_ID_0));

    // Disable flash write/erase operations
    PLIB_NVM_MemoryModifyInhibit(NVM_ID_0);
} // NVMPageErase
```

## CONVERSION ADRESSE VIRTUELLE EN PHYSIQUE

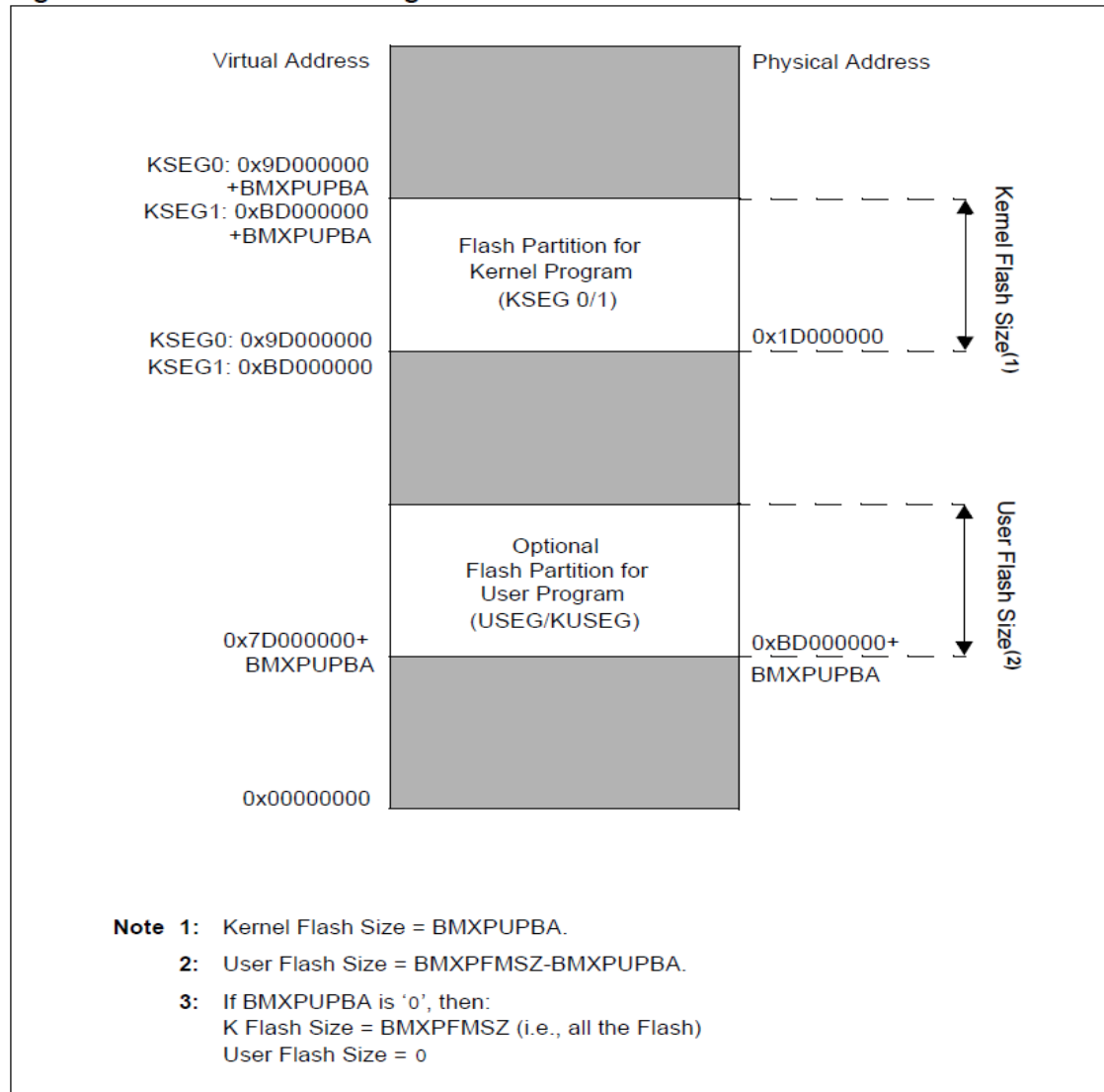
Le contrôleur de la flash travaille avec les adresses physiques alors que le compilateur utilise les adresses virtuelles.

```
// Converts a virtual memory address to a physical one
uint32_t virtualToPhysical(uint32_t address)
{
    return (address & 0x1FFFFFFF);
}
```

## MÉMOIRE PROGRAMME DU PIC32MX795F512L

Voici l'organisation de la mémoire flash du PIC32MX.

**Figure 3-3: Flash Partitioning**



En utilisant le debugger, on peut observer la situation du programme (sous Windows > PIC Memory Views). On peut en conclure que le programme se trouve dans la partie Kernel. (Adresse virtuelle 9D00\_xxxx)

| Flash Memory |      |           |          |       |                           |
|--------------|------|-----------|----------|-------|---------------------------|
|              | Line | Address   | Opcode   | Label | DisAssy                   |
|              | 1100 | 9D00_112C | 24050073 |       | ADDIU A1, ZERO, 115       |
|              | 1101 | 9D00_1130 | 24050063 |       | ADDIU A1, ZERO, 99        |
|              | 1102 | 9D00_1134 | 10850153 |       | BEQ A0, A1, 0x1D001684    |
|              | 1103 | 9D00_1138 | 28850064 |       | SLTI A1, A0, 100          |
|              | 1104 | 9D00_113C | 50A00128 |       | BEQL A1, ZERO, 0x1D0015E0 |
|              | 1105 | 9D00_1140 | 24050064 |       | ADDIU A1, ZERO, 100       |
|              | 1106 | 9D00_1144 | 1480016D |       | BNE A0, ZERO, 0x1D0016FC  |
|              | 1107 | 9D00_1148 | 24050058 |       | ADDIU A1, ZERO, 88        |
|              | 1108 | 9D00_114C | 8FBF0054 |       | LW RA, 84(SP)             |
|              | 1109 | 9D00_1150 | 02E01021 |       | ADDU V0, S7, ZERO         |
|              | 1110 | 9D00_1154 | 8FBF0050 |       | LW S8, 80(SP)             |
|              | 1111 | 9D00_1158 | 8FB7004C |       | LW S7, 76(SP)             |
|              | 1112 | 9D00_115C | 8FB60048 |       | LW S6, 72(SP)             |

## POSITIONNEMENT DANS LA MÉMOIRE

Il est assez difficile de trouver des informations sur la zone utilisée par le programme réalisé.

Avec Harmony, il n'y a pas de fonctions permettant de connaître les tailles de page et de row, nous les obtenons au travers de l'exemple :

```
/* Row size for pic32mx795 device is 512 bytes */
#define DEVICE_ROW_SIZE_DIVIDED_BY_4          128

/* Page size for pic32mx795 device is 4 Kbytes */
#define DEVICE_PAGE_SIZE_DIVIDED_BY_4        1024
```

## SOLUTION AVEC ALLOCATION

Sur la base de l'exemple `dee_emulation_PIC32` fourni par Microchip (avant introduction Harmony), avec la déclaration suivante :

```
// Row dans flash pour data
const uint32_t eedata_addr[DEVICE_ROW_SIZE_DIVIDED_BY_4]
    __attribute__((aligned(4096), space(prog))) ;

#define NVM_PROGRAM_PAGE ((uint32_t)&eedata_addr[0])
```

😊 En réservant une zone, on évite les risques de conflit et on obtient la possibilité d'utiliser le debugger.

Notre page de 4096 octets se trouve placée à l'adresse 0x9D00'2000 entre 2 parties du programme.

On peut vérifier cela avec l'observation de la mémoire :

Macro Expansion

Output

Tasks

Search Re

Le programme commence à l'adresse virtuelle 0x9D00\_0000.

Macro Expansion

Output

Tasks

Search R

| Line  | Address   | Opcode   | Label    | DisAssy             |
|-------|-----------|----------|----------|---------------------|
| 32769 |           | 5974D476 |          | BLEZL T3, 0x151D8   |
|       |           |          |          | Program Memory      |
| 32771 | 9D00_0000 | 27BDF98  | vfprintf | ADDIU SP, SP, -104  |
| 32772 | 9D00_0004 | 00001021 |          | ADDU V0, ZERO, ZERO |
| 32773 | 9D00_0008 | AFB3004C |          | SW S3, 76(SP)       |
| 32774 | 9D00_000C | AFB10044 |          | SW S1, 68(SP)       |
| 32775 | 9D00_0010 | AFB00040 |          | SW S0, 64(SP)       |
| 32776 | 9D00_0014 | AFBF0064 |          | SW RA, 100(SP)      |
| 32777 | 9D00_0018 | AFBE0060 |          | SW S8, 96(SP)       |
| 32778 | 9D00_001C | AFB7005C |          | SW S7, 92(SP)       |
| 32779 | 9D00_0020 | AFB60058 |          | SW S6, 88(SP)       |
| 32780 | 9D00_0024 | AFB50054 |          | SW S5, 84(SP)       |
| 32781 | 9D00_0028 | AFB40050 |          | SW S4, 80(SP)       |






Memory

Execution Memory

Format

Code

Il continue après notre zone qui va de 9D00\_2000 à 9D00\_2FFF

| Macro Expansion  |       | Output    |          | Tasks          | Search R              |
|--|-------|-----------|----------|----------------|-----------------------|
|   | Line  | Address   | Opcode   | Label          | DisAssy               |
|   | 35838 | 9D00_2FEC | 00000000 |                | NOP                   |
|   | 35839 | 9D00_2FF0 | 00000000 |                | NOP                   |
|   | 35840 | 9D00_2FF4 | 00000000 |                | NOP                   |
|  | 35841 | 9D00_2FF8 | 00000000 |                | NOP                   |
|  | 35842 | 9D00_2FFC | 00000000 |                | NOP                   |
|  | 35843 | 9D00_3000 | 27BDFFE8 | BSP_Initialize | ADDIU SP, SP, -24     |
|  | 35844 | 9D00_3004 | AFBF0014 |                | SW RA, 20(SP)         |
|  | 35845 | 9D00_3008 | 0F401BCD |                | JAL SYS_DEVCON_JTA... |
|  | 35846 | 9D00_300C | 00000000 |                | NOP                   |
|  | 35847 | 9D00_3010 | 3C04BF88 |                | LUI A0, -16504        |
|  | 35848 | 9D00_3014 | 8C836040 |                | LW V1, 24640(A0)      |
|  | 35849 | 9D00_3018 | 24020001 |                | ADDIU V0, ZERO, 1     |
|  | 35850 | 9D00_301C | 7C430004 |                | INS V1. V0. 0. 1      |

Memory 

Execution Memory ▼

 Format 

Code ▼

## EXEMPLE ÉCRITURE ET RELECTURE D'UNE STRUCTURE

Voici un exemple montrant les actions nécessaires pour écrire le contenu d'une structure et la relire.

### ACTIONS D'ÉCRITURE D'UNE STRUCTURE

L'action est réalisée par la fonction `NVM_WriteBlock`. Cette fonction réalise la séquence suivante :

- Effacement de la page correspondant à `eedata_addr`
- Copie de la structure dans `databuff` (image row en RAM)
- Ecriture d'un row avec le contenu de `databuff`

```
// Cette fonction écrit un bloc de data au début de la zone flash
// PData correspond à l'adresse du bloc de donnée
// DataSize est la taille en octets du bloc de donnée
```

```
void NVM_WriteBlock(uint32_t *pData, uint32_t DataSize)
{
    int i, iMax;
    uint32_t destAddr = NVM_PROGRAM_PAGE;
    // Efface la page dans la flash
    NVMpageErase(NVM_PROGRAM_PAGE) ;
    if ( (DataSize % 4) != 0 ) {
        iMax = (DataSize / 4) + 1;
    } else {
        iMax = DataSize / 4 ;
    }

    // Copie le bloc dans databuff
    for ( i = 0 ; i < iMax; i++ ) {
        databuff[i] = *pData;
        pData++;
    }

    NVMwriteRow(NVM_PROGRAM_PAGE, DATA_BUFFER_START) ;
}
```

### LA FONCTION NVMWRITEROW

Cette fonction reprise de l'exemple permet d'écrire le contenu d'un tampon en RAM dans la flash.

```
/*
    Function:
        void NVMwriteRow(uint32_t address,
                        uint32_t dataAddress)

    Summary:
        Writes a row in flash memory (1KB)
*/
```



```
void NVMwriteRow(uint32_t destAddr, uint32_t srcAddr)
{
    // Base address of row to be written to (destination)
    PLIB_NVM_FlashAddressToModify(NVM_ID_0,
                                    virtualToPhysical(destAddr));

    // Data buffer address (source)
    PLIB_NVM_DataBlockSourceAddress(NVM_ID_0,
                                    virtualToPhysical(srcAddr));

    // Disable flash write/erase operations
    PLIB_NVM_MemoryModifyInhibit(NVM_ID_0);

    // Select row write function & enable
    // flash write/erase operations
    PLIB_NVM_MemoryOperationSelect(NVM_ID_0,
                                    ROW_PROGRAM_OPERATION);

    // Allow memory modifications
    PLIB_NVM_MemoryModifyEnable(NVM_ID_0);

    // Write the unlock key sequence
    PLIB_NVM_FlashWriteKeySequence(NVM_ID_0, 0xAA996655);
    PLIB_NVM_FlashWriteKeySequence(NVM_ID_0, 0x556699AA);

    // Start the operation
    PLIB_NVM_FlashWriteStart(NVM_ID_0);

    // Attente fin de l'écriture
    while (!PLIB_NVM_FlashWriteCycleHasCompleted(NVM_ID_0));
}
```

## LECTURE D'UNE STRUCTURE

En principe il est possible de lire directement la mémoire flash. Pour éviter d'accéder n'importe comment, la fonction `NVM_ReadBlock` lit le début de la zone `eedata_addr` et la copie à l'adresse de destination.

```
void NVM_ReadBlock(uint32_t *pData, uint32_t DataSize)
{
    int i, iMax;

    if ( (DataSize % 4) != 0 ) {
        iMax = (DataSize / 4) + 1;
    } else {
        iMax = DataSize / 4 ;
    }
    for ( i = 0 ; i < iMax; i++ ) {
        *pData = eedata_addr[i];
        pData++;
    }
} // NVM_ReadBlock
```

## EXEMPLE UTILISATION ET TEST DU RÉSULTAT

Pour vérifier si le contenu écrit dans la flash est correct, on compare le contenu des deux structures.

Déclaration dans l'application :

```
S_ParamGen LocalParamGen;  
S_ParamGen ReadParamGen;
```

Action dans l'application :

```
// Enregistrement des paramètres  
LocalParamGen.Forme = SignalSinus;  
LocalParamGen.Frequence = 20;  
LocalParamGen.Amplitude = 6789;  
LocalParamGen.Offset = -1234;  
LocalParamGen.Duty = 75;  
LocalParamGen.Magic = 0x12345678;  
NVM_WriteBlock((uint32_t*)&LocalParamGen,  
                sizeof(LocalParamGen)) ;  
  
// Relecture des paramètres  
NVM_ReadBlock((uint32_t*)&ReadParamGen,  
               sizeof(ReadParamGen)) ;  
  
// Test si match  
if (ReadParamGen.Forme == LocalParamGen.Forme &&  
    ReadParamGen.Frequence == LocalParamGen.Frequence &&  
    ReadParamGen.Amplitude == LocalParamGen.Amplitude &&  
    ReadParamGen.Offset == LocalParamGen.Offset &&  
    ReadParamGen.Duty == LocalParamGen.Duty &&  
    ReadParamGen.Magic == LocalParamGen.Magic ) {  
    lcd_gotoxy(1,3);  
    printf_lcd("Param OK          " );  
} else {  
    lcd_gotoxy(1,3);  
    printf_lcd("Param mismatch" );  
}
```

☺ On obtient l'affichage de Param OK.

## UTILITAIRE À DISPOSITION

Les fichiers Mc32NVMUtil.h et Mc32NVMUtil.c sont disponibles sous :

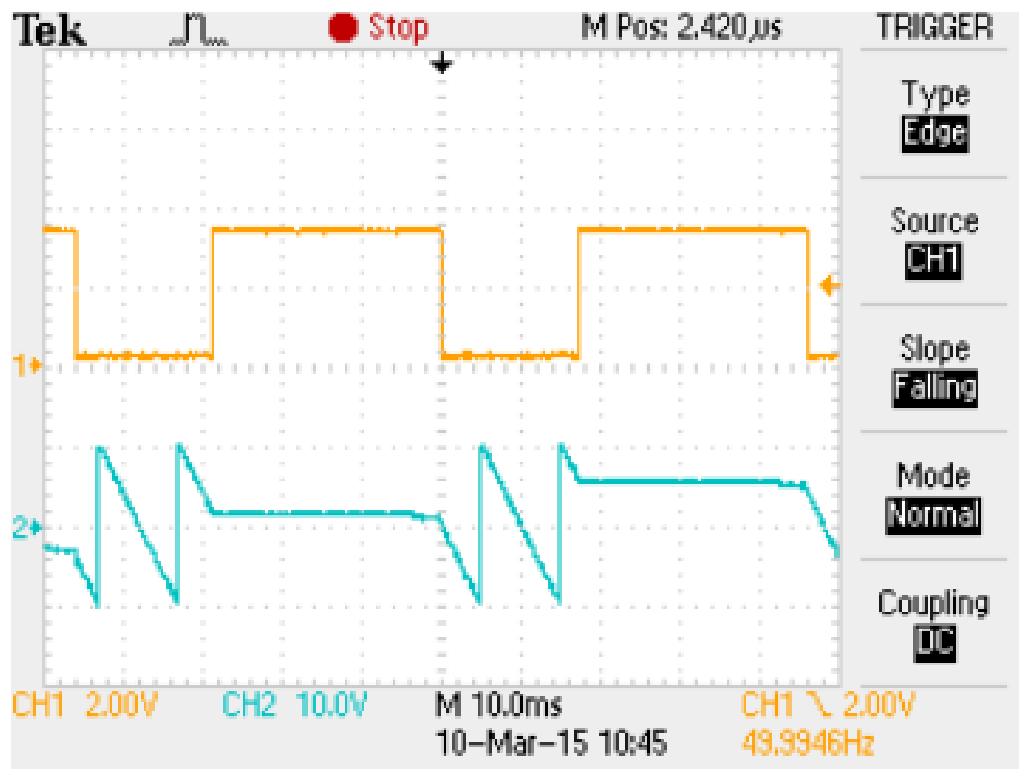
...Maitres-Eleves\SLO\Modules\SL229\_MINF\PIC32MX\_Utilitaires(PlibHarmony)\NVM

## MESURE EN FONCTIONNEMENT DYNAMIQUE

Activation de l'application toutes les 20 ms et enregistrement des paramètres à chaque cycle durant 300 cycles.

Mesure avec canal 1 = LED\_2

Mesure avec canal 2 = sortie DAC (signal dent de scie)



TDS 2024C - 10:38:51 10.03.2015

On constate que durant la sauvegarde (signal LED\_2 au niveau haut), on n'a plus de signal en sortie du DAC. La sauvegarde dure approximativement 30 ms. Le timer qui déclenche l'application toutes les 20 ms semble figé.

☹ Il y a une situation de blackout durant l'action avec la flash programme.

## CARACTÉRISTIQUES DE LA MÉMOIRE PROGRAMME

**TABLE 31-11: DC CHARACTERISTICS: PROGRAM MEMORY<sup>(3)</sup>**

| DC CHARACTERISTICS |        |                             | Standard Operating Conditions: 2.3V to 3.6V<br>(unless otherwise stated)<br>Operating temperature $-40^{\circ}\text{C} \leq T_A \leq +85^{\circ}\text{C}$ for Industrial |                        |      |       |   |
|--------------------|--------|-----------------------------|--|------------------------|------|-------|---|
| Param. No.         | Symbol | Characteristics             | Min.   | Typical <sup>(1)</sup> | Max. | Units | Conditions                                    |
|                    |        | <b>Program Flash Memory</b> |  |                        |      |       |   |
| D130               | EP     | Cell Endurance              | 1000   | —                      | —    | E/W   | —   |
| D130a              | EP     | Cell Endurance              | 20,000   | —                      | —    | E/W   | <b>Note 4</b>                                 |
| D131               | VPR    | VDD for Read                | 2.3  | —                      | 3.6  | V     | —   |
| D132               | VPEW   | VDD for Erase or Write      | 3.0  | —                      | 3.6  | V     | —   |
| D132a              | VPEW   | VDD for Erase or Write      | 2.3  | —                      | 3.6  | V     | <b>Note 4</b>                                 |
| D134               | TRETD  | Characteristic Retention    | 20   | —                      | —    | Year  | Provided no other specifications are violated |

4: This parameter applies to PIC32MX534/564/664/764 devices only. This information is preliminary.

**TABLE 31-11: DC CHARACTERISTICS: PROGRAM MEMORY<sup>(3)</sup> (CONTINUED)**

| DC CHARACTERISTICS |        |   | Standard Operating Conditions: 2.3V to 3.6V<br>(unless otherwise stated)<br>Operating temperature $-40^{\circ}\text{C} \leq T_A \leq +85^{\circ}\text{C}$ for Industrial |                        |      |               |            |
|--------------------|--------|---|--|------------------------|------|---------------|------------|
| Param. No.         | Symbol | Characteristics   | Min.   | Typical <sup>(1)</sup> | Max. | Units         | Conditions |
| D135               | IDDP   | Supply Current during Programming                       | —  | 10                     | —    | mA            | —          |
| D136               | TWW    | Word Write Cycle Time                                   | 20   | —                      | 40   | $\mu\text{s}$ | —          |
|                    | TRW    | Row Write Cycle Time<br>(Note 2)<br>(128 words per row) | 3  | 4.5                    | —    | ms            | —          |
| D137               | TPE    | Page Erase Cycle Time                                   | 20   | —                      | —    | ms            | —          |
|                    | TCE    | Chip Erase Cycle Time                                   | 80   | —                      | —    | ms            | —          |

Du tableau ci-dessus, on retire :

- 1000 cycles d'effacement (minimum garanti)
- Rétention min de 20 ans
- Durée d'une écriture ou de l'effacement d'un bloc (row) 4.5 ms (valeur typique).
- Durée d'une écriture ou de l'effacement d'une page 20 ms (valeur minimum).

## CONCLUSION

Sur la base des informations fournies dans ce document, les étudiants disposent des éléments nécessaires pour réaliser l'enregistrement et la lecture de données dans la mémoire flash du programme.

## **HISTORIQUE DES VERSIONS**

### **VERSION 1.0 FÉVRIER 2016**

Document original. Création avec Harmony v1.06.

### **VERSION 1.1 FÉVRIER 2021**

Ajout attente fin d'opération à la fin des fonctions NVMpageErase et NVMwriteRow.  
Test sous Harmony 2.06.