Application Note

NVDS Operation in the Z8 Encore! XP® F042A Series Microcontroller

AN029301-1008

Z8 Encore! XP® Flash Microcontrollers

Introduction

Zilog's Z8 Encore! XP® F042A Series microcontroller unit (MCU) features a Non-Volatile Data Storage (NVDS) memory, that can accommodate up to 128 bytes of data. The NVDS read and write cycle times vary over a number of write operations, and are therefore critical to the application. This Application Note describes the read and write operations on the Z8 Encore! XP NVDS.

NVDS Operations

The NVDS memory on the Z8 Encore! XP F042A Series MCU functions as an Electrically Erasable Programmable Read-Only Memory (EEPROM) emulated within the Flash Memory. You can perform over 100,000 write cycles on this NVDS memory.

By default, the Z8 Encore! XP F042A Series MCU uses the Internal Precision Oscillator (IPO) at 5.52960 MHz as the system clock. The NVDS read and write cycle times listed in this Application Note are based on the IPO usage.

This section discusses the following topics in detail:

- Interrupt Handling During NVDS Operations
- NVDS Write Operation
- NVDS Read Operation
- Power Failure Protection
- Optimizing NVDS Memory Usage for Execution Speed

The following two routines are required to access the Z8 Encore! XP NVDS:

1. Write routine
2. Read routine

These routines are accessed with a `CALL` instruction to a predefined address outside the user-accessible program memory area. The NVDS address and the data are single-byte values. Because the write and read routines modify the working register set, the code must preserve the required register values. This is done by pushing these values onto the stack or by changing the working register pointer, prior to NVDS read and write operations.

When the Z8 Encore! XP F042A Series device is in the DEBUG Mode, you can view the NVDS memory locations by performing the following steps in ZDS II:

1. Navigate to **View** → **Debug Windows** → **Memory**. The **Memory** Window appears.
2. In the **Memory** Window, select the **NVDS** option from the **Space** drop down menu, to view the NVDS contents.

Interrupt Handling During NVDS Operations

By default, interrupts are not disabled during NVDS read and write operations. For correct NVDS operation, disabling the interrupts is recommended, to ensure that interrupts do not modify the working register and stack contents. NVDS requires a stack of 15 bytes size. The contents of the working register set are overwritten during NVDS read and write operations. For correct NVDS operation, the Flash Frequency Registers must be programmed, based on the system clock

frequency. For detailed information about the Flash Frequency Registers, refer to *Z8 Encore! XP® F082A Series Product Specification (PS0228)*.

NVDS Write Operation

To write a byte of data to an NVDS memory location, the code must first push the address, then the data byte onto the stack. The code issues a `CALL` instruction to the address of the byte write routine (`0x10B3`). At the return from the sub-routine, the write status byte resides in working register R0. The bit fields of this status byte are listed in Table 1 on page 3. The contents of the status byte are undefined for write operations to illegal addresses. Also, code should pop the address and data bytes off the stack. The write routine uses 13 bytes of the stack, in addition to the 2 bytes of address and data, pushed by you. Ensure that sufficient memory is available for this stack usage.

The following in line assembly code snippet illustrates an NVDS write operation:

```
void NVDS_Write(unsigned char
NVDS_Address,unsigned char
NVDS_Write_Data)
{
    asm("\tLDX  R2,
    _NVDS_Address");
    asm("\tPUSH R2");
    asm("\tLDX  R2,
    _NVDS_Write_Data");
    asm("\tPUSH R2");
    asm("\tCALL %10B3");
    asm("\tPOP  R2");
    asm("\tPOP  R2");
}
```

ZDS II provides the `WRITE_NVDS()` API library for writing to the NVDS memory locations. This API internally executes the CALL to the byte write routine. The prototype of the `WRITE_NVDS()` API is defined in the `ez8.h` header file:

```
reentrant int WRITE_NVDS (char value,
char address)
```

Where,

- `value` represents the data to be written
- `address` represents the NVDS memory location, to which the data must be written (`0x00` to `0x7F` for a 128 byte NVDS)

If successful, the `WRITE_NVDS()` function returns a 0; otherwise, the function returns a non-zero value. Include the `ez8.h` header file in the code to use the `WRITE_NVDS()` API.

The execution time for the NVDS write operation is non-uniform because of the Flash Memory architecture. Generally, a minimal time of 400 µs is required to write a single byte of data to the NVDS. A maintenance operation is internally performed after every 400 or 500 write cycles. Faster system clock speed results in proportionally lower execution time. NVDS write operations to addresses other than the specified locations (`0x00` to `0x7F` for a 128 byte NVDS) are illegal. Such illegal write operations execute in 10.8 µs.

Table 1. Write Status Byte

Bits	7	6	5	4	3	2	1	0
Field	Reserved				RCPY	PF	AWE	DWE
Default Value	0	0	0	0	0	0	0	0

Bit Position	Description
Reserved	These bits must be 0.
RCPY (Recopy Subroutine Executed)	A recopy subroutine is executed. These operations require significantly longer time than a normal write operation.
PF (Power Failure Indicator)	A power failure or system reset occurred during the most recent attempted write operation to the NVDS memory location.
AWE (Address Write Error)	An address byte failure occurred during the most recent attempted write operation to the NVDS memory location.
DWE (Data Write Error)	A data byte failure occurred during the most recent attempted write operation to the NVDS memory location.

The following code snippet illustrates an NVDS write operation to consecutive locations.

```
#include <ez8.h>
#define number_of_bytes 16 // Can be
//maximum 128 bytes depending on
//device.
char NVDS_Address = 0x00;
char NVDS_Write_Data=1;
int write_status;
extern unsigned char _intrp;
main()
{
  for (i=0;i< number_of_bytes;i++)
    {
    DI();
    asm("pushx rp"); // Save contents
    //of Register Pointer
    _intrp=_intrp; // needed to use in
    //inline assembly.
    asm("LDX RP, __intrp"); // RP
    //points to different register
    //bank.
    write_status=WRITE_NVDS
    (NVDS_Write_Data,NVDS_Address);
    //Write NVDS_Write_Data to
    //NVDS_Address.
    if(write_status!=0x00)
    //If unsuccessful take action
    //here.
    //{----------------------------}
    asm("popx rp"); // Restore RP.
    NVDS_Address++;
    NVDS_Write_Data++;
    EI();
    }
}
```

Table 2 lists the time required for writing data of different sizes to the NVDS memory location in the first write attempt

Table 2. NVDS Write Time

Data Size (bytes)	Write Time (sec)
1	400 µs
16	6.8 ms
64	27.2 ms

Table 3 lists the time required for writing a single byte of data to the NVDS memory location in different write attempts.

Table 3. NVDS Write Time for a Single Byte

Write Attempt (bytes)	Write Time (µs)
1	400
150	406
210	412

Note: *The register pointer (RP) must point to a different register bank, prior to NVDS operation, to avoid modifying the working register contents. To point the RP to a different register bank, load the* `_intrp` *byte to RP. The* `_intrp` *byte is a single byte, defined in the standard startup file, and is primarily used to point RP to a different register bank during an interrupt. The usage of* `_intrp` *is valid only if interrupts are disabled. If interrupts are enabled during NVDS operation, be careful in handling the working register set.*

NVDS Read Operation

To read a byte of data from an NVDS memory location, the code must first push the address onto the stack. The code issues a `CALL` instruction to the address of the byte read routine (`0x1000`). At the return from the sub-routine, the read byte resides in working register R0, and the read status byte resides in working register R1. The contents of the status byte are undefined for read operations to illegal addresses. The status byte returned is zero for a successful read operation, and is determined by a Cyclic Redundancy Check (CRC). If the status byte for the read operation is a non-zero value, then the NVDS memory location being read contains a corrupted value. In this case the value returned through the R0 working register is the byte recently written to the NVDS memory location that does not exhibit CRC error. Also, the code should pop the address byte off the stack. The read routine uses 9 bytes of stack space in addition to the 1 byte of address pushed by you.

The following assembly code snippet illustrates an NVDS read operation:

```
char NVDS_Read (unsigned char
NVDS_Address)
{
    asm("\tLDX  R2, _NVDS_Address");
    asm("\tPUSH R2");
    asm("\tCALL %1000");
    asm("\tPOP  R2");
}
```

ZDS II provides the `READ_NVDS()` API library for reading data from the NVDS memory location. The prototype of the `READ_NVDS()` API is defined in the `ez8.h` header file:

```
reentrant char READ_NVDS(char
address)
```

Where,

`address` represents the NVDS memory location from which the data must be read.

On successful execution, the `READ_NVDS()` API returns the data read from the specified NVDS memory location. Include the `ez8.h` header file in the code to use the `READ_NVDS()` API.

The execution time for the NVDS read operation is non uniform because of the Flash Memory architecture. A single-byte read operation requires a minimal time of 160 µs. Faster system clock speed results in proportionally lower execution time. NVDS read operations from addresses other than the specified locations are illegal. Such illegal read operations execute in 10.8 µs.

The following code snippet illustrates an NVDS read operation to consecutive locations:

```
#include <ez8.h>
#define number_of_bytes 16 // Can be
// maximum 128 bytes depending on
//device.
char NVDS_Address = 0x00;
char NVDS_Read_Data;
extern unsigned char _intrp;
main()
{
    for (i=0;i< number_of_bytes;i++)
    {
     DI();
    asm("pushx rp"); // Save contents
    //of Register Pointer
    _intrp=_intrp; // needed to use in
    //inline assembly.
    asm("LDX RP, _intrp"); // RP
    //points to different register
    //bank.
    NVDS_Read_Data=
    READ_NVDS(NVDS_Address)
    //NVDS_Read_Data holds
    //the read data
    asm("popx rp"); // Restore RP.
    NVDS_Address++;
    EI();
     }
}
```

Note: *From Table 4 it is clear that the NVDS read times vary, minimum being 160 µs, after multiple NVDS write operations to locations other than N. The NVDS read and write cycle times depend on the written address, the order of the written address, and the number of times a particular address is written. There is no specific way by which you can predict the read and write time under normal conditions.*

Power Failure Protection

The Z8 Encore! XP® NVDS feature employs protective mechanisms to ensure that a power failure endangers only the most recently written byte. Bytes previously written to the NVDS memory location are not modified. For this protection feature to function, the Voltage Brownout (VBO) feature must be enabled and configured for a threshold voltage of 2.4 V or above (Refer to the *Low-Power Modes* and *Trim Bit Address Space* sections of *Z8 Encore! XP® F082A Series Product Specification (PS0228)*).

If a system reset (such as a pin reset or Watchdog Timer (WDT) reset) occurs during a write operation, the value of the currently written byte is unknown, but none of the other bytes are modified.

Optimizing NVDS Memory Usage for Execution Speed

Table 4 lists the time required for reading a single byte of data from a fixed NVDS memory location. It is clear that NVDS read times vary and are at minimum 160 µs. The time required for reading data from address N is a function of the number of writes to addresses; other than N after the most recent write to N. Additionally, the read time depends on the number of writes, after the most recent page erase during a maintenance operation. These observations indicate that every write after the most recent page erase causes the read times of unwritten addresses to increase by 4 µs.

Table 4. NVDS Read Time

Write Attempt (bytes)	Read Time for Location N (µs)
1	160
2	164
3	168
4	172
5	176
6	180
7	184
8	188
9	192
10	196
11	200

Table 4. NVDS Read Time (Continued)

Write Attempt (bytes)	Read Time for Location N (µs)
12	204
13	208
14	212
15	216
16	220
17	224
18	228
19	232
20	236

If the NVDS read performance is critical to software architecture, the following conventions can be followed to optimize the code for speed:

- Periodically refresh all addresses that are used, by rotating the writes evenly among all addresses intended to be used. This operation in turn brings all of the reads closer to the minimum read time. Because the minimum read time is much less than the write time, the actual speed benefits are not always realized.
- Use as few unique addresses as possible, to optimize the impact of refreshing, and minimize the requirement for refreshing the addresses.

Summary

This Application Note describes the read and write operations on the Z8 Encore! XP® NVDS. The minimum NVDS read time for a single byte of data is around 160 µs. The minimum NVDS write time for a single byte of data is around 400 µs, with 5.5296 MHz IPO, as system clock. Faster system clock speed results in proportionally lower execution time. The NVDS read times vary drastically compared to the NVDS write times. Therefore, the application must be designed considering the variable read times. Fixed NVDS memory locations can be used to implement the read and write routines in the application. The Z8 Encore! XP NVDS memory can be used in applications where data must be logged, updated regularly, and preserved on power off.

Warning: DO NOT USE IN LIFE SUPPORT

LIFE SUPPORT POLICY

ZILOG'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS PRIOR WRITTEN APPROVAL OF THE PRESIDENT AND GENERAL COUNSEL OF ZILOG CORPORATION.

As used herein

Life support devices or systems are devices which (a) are intended for surgical implant into the body, or (b) support or sustain life and whose failure to perform when properly used in accordance with instructions for use provided in the labeling can be reasonably expected to result in a significant injury to the user. A critical component is any component in a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system or to affect its safety or effectiveness.

Document Disclaimer