AN1015
Application note

Software techniques for improving microcontrollers EMC performance

Introduction

A major contributor to improved EMC performance in microcontroller-based electronics systems is the design of hardened software.

Problems induced by EMC disturbances need to be considered as early as possible in the design phase. EMC-oriented software increases the security and the reliability of your application. EMC-hardened software is inexpensive to implement, improves the final goods immunity performance and saves hardware and development costs. You should consider EMC disturbances to analog or digital data just like any other application parameter.

Examples of problems induced by EMC disturbances:

- Microcontroller not responding
- Program Counter runaway
- Execution of unexpected instructions
- Bad address pointing
- Bad execution of subroutines
- Parasitic reset and/or parasitic interrupts
- Corruption of IP configuration
- I/O deprogramming

Examples of consequences of failing software:

- Unexpected response of product
- Loss of context
- Unexpected branch in process
- Loss of interrupts
- Loss of data integrity
- Corrupted reading of input values.

This application note deals with two categories of software techniques, namely:

- Preventive techniques: these can be implemented in existing designs, their purpose is to improve product robustness.
- Auto-recovery techniques: when a runaway condition is detected, a recovery subroutine is used to take the decision to execute fail-safe routines, optionally sending a warning and then automatically returning back to normal operations (this operation may be absolutely transparent to the user of the application).

Contents

List of tables

List of figures

1 Related documents

- AN3181 “Guidelines for obtaining IEC 60335 Class B certification in an STM8 application”
- AN3307 “Guidelines for obtaining IEC 60335 Class B certification for any STM32 application”
- AN4435 “Guidelines for obtaining UL/CSA/IEC 60335 Class B certification in any STM32 application”

2 Preventive techniques

You can implement preventive techniques in existing designs to improve product robustness and immunity against external or internal EMC disturbance.

2.1 Using the watchdog and time control techniques

The watchdog is the most efficient tool available to ensuring that the MCU can recover from software runaway failures. Its principle is very simple: it is a timer which generates an MCU reset at the end of count. Once the watchdog is started, the only way of preventing the watchdog from resetting the microcontroller is to update the counter periodically in the program.

But to make the watchdog work at its full potential, you have to insert the enable and refresh instructions in your software at the right place of code execution.

Figure 1 shows two classic examples of bad watchdog implementation.

To do it the right way (see *Figure 2*), the following rules should be implemented:

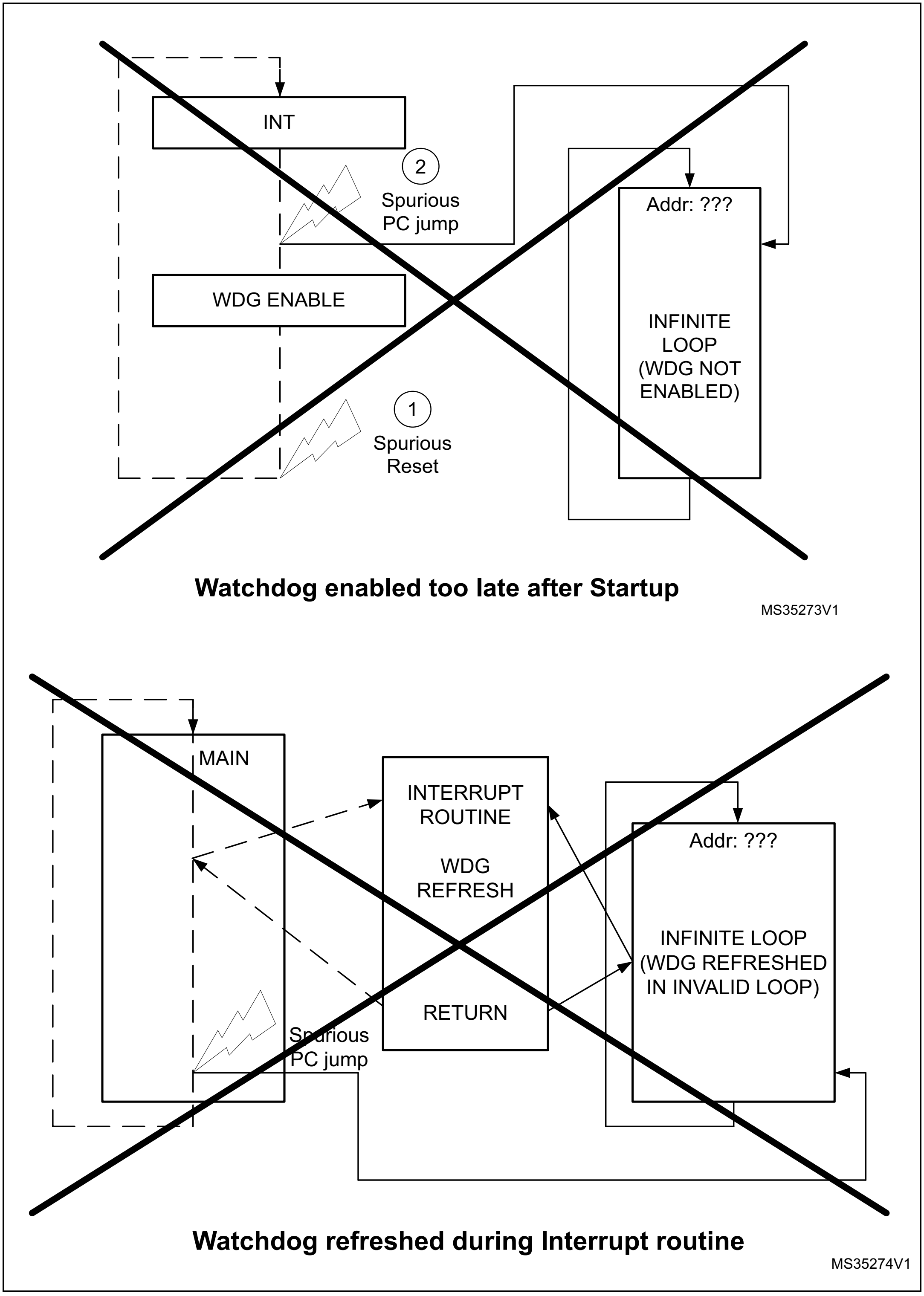
- Enable the watchdog as soon as possible after reset, or use the Hardware option if available.
- Never refresh the watchdog during an interrupt routine or inside any local loop not guarded by timeout at code.

It is very important to optimize the period between the two refresh instructions according to the duration of the various routines, including the interrupt routines.

The minimum use of the watchdog resets the MCU, this means that the program execution context is lost as well as the application data's integrity.

After reset, in addition to enabling the watchdog, on some MCUs you can use the reset flags to distinguish between Power On or Low Voltage reset or watchdog reset (refer to *Section 3.3: Using the reset flags to identify the reset source*. for more details).

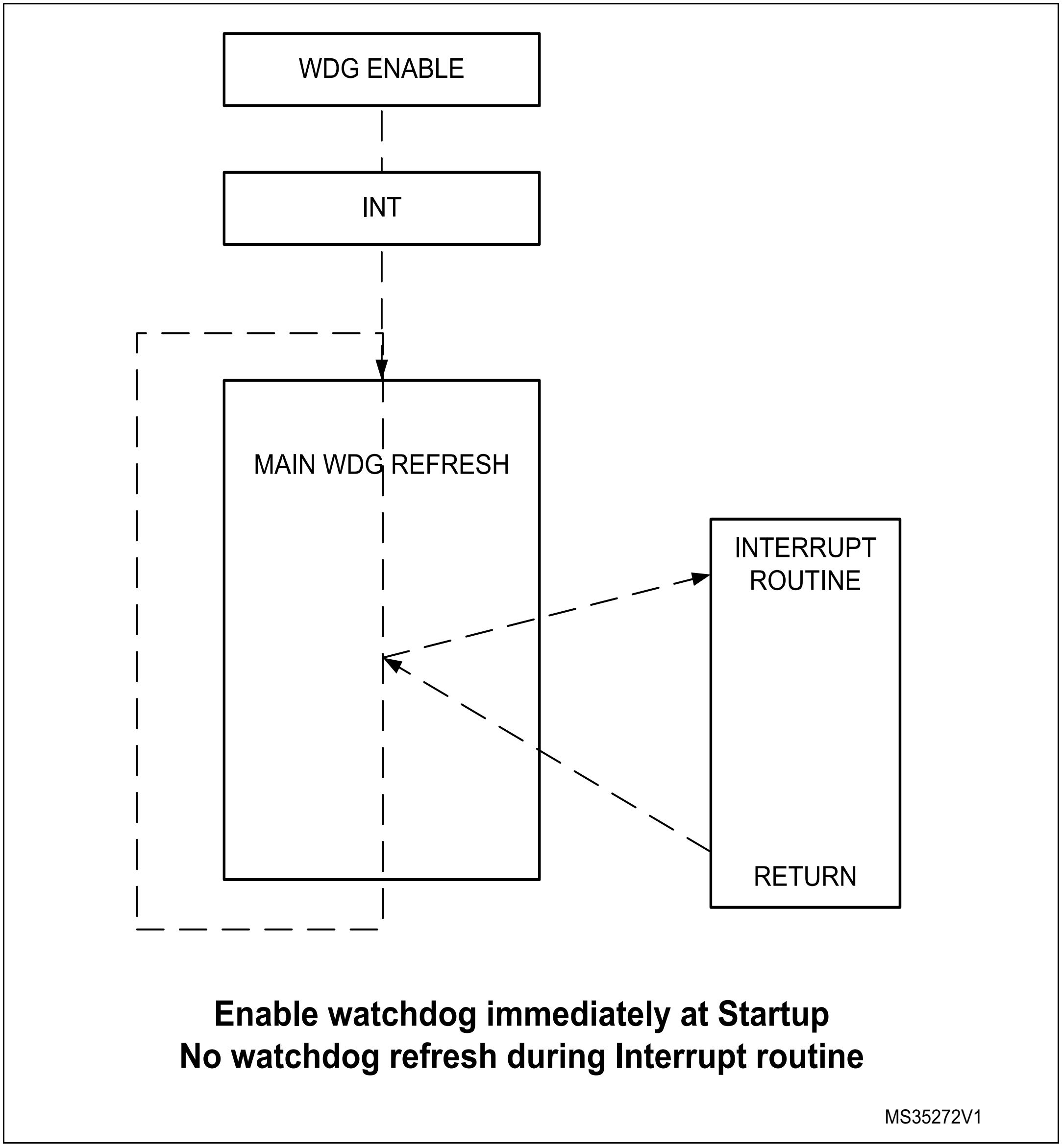
Figure 1. Classic examples of bad watchdog usage



Watchdog enabled too late after Startup

Watchdog refreshed during Interrupt routine

Figure 2. Example of correct watchdog usage


WDG ENABLE
INT
MAIN WDG REFRESH
INTERRUPT ROUTINE
RETURN
Enable watchdog immediately at Startup
No watchdog refresh during Interrupt routine
MS35272V1

2.2 Securing the unused program memory area

In most applications, program memory space is not filled completely by user code. For extra security, fill the unused memory locations with code that forces a watchdog reset or jumps to a known program location if you do not want to generate a reset.

This will ensure that even if the program counter is corrupted and jumps to an unused memory location, the MCU will recover and return to normal operations.

In this unused area you can also jump to a recovery fail safe subroutine, which allows you to return to normal operations.

For STM8 users, the STM8 "TRAP" instruction is also very convenient (only one instruction byte:83) for generating a software interrupt in order to recover from a jump to an unexpected location in memory.

Another effective way for recovery to normal operation is filling memory by value of illegal instruction opcode which fetch and execution generates reset on STM8 microcontrollers.

STM32 microcontrollers with ARM® Cortex®-M core use fault exception which trap illegal memory accesses and illegal program behavior which may occure if the system is exposed to EMC disturbances. The undefined instruction opcode may be used to fill the unused memory of STM32 microcontroller to rise the Usage Fault exception where the fail safe routine recovers from errors in case of program counter run-away.

Another option is to use System serivce call with SVC instruciton to execute fail safe routine.

2.3 Input filtering and comparison

The routine which checks several times that that the state of input pin is stable before validating the state and continuing the program execution is a good practice to avoid unwanted reaction on spikes caused by external noise induced in input circuit.

This is a simple means of critical input filtering for no extra cost!

2.4 Management of unused interrupt vectors

To avoid problems caused by unexpected interrupt occurrences (whatever the source) it is recommended to manage all the possible interrupt sources by putting a valid interrupt routine address in the corresponding vector.

In the example below the unused interrupt vectors point to a fault management routine label filled with a simple “return from interrupt” instruction.

2.5 Removing illegal and critical bytes from your code

2.5.1 Critical Bytes

A critical byte is an instruction switching MCU in low power modes which is decoded by the microcontroller and forces it to stop executing any further instructions.

When the PC is corrupted it often becomes desynchronized (as most of the instructions have several bytes), and as a result it may read and decode critical bytes.

To check and minimize the occurrence of these critical bytes you can check the program ".list" file.

Very often critical bytes are generated by the compiler as label address bytes. In this case, if you simply insert one or several NOP instructions, all the label addresses will shift and this will change the critical byte value to another value.

2.5.2 Illegal Bytes

Illegal bytes are defined as any byte value which is not part of the instruction set. They will either be executed as a NOP instruction or (on some MCUs) a reset is generated if an illegal byte is encountered. In this case, use the techniques described above (for critical bytes) to remove illegal bytes from your code.

2.6 Averaging the A/D converter results

If you are performing A/D conversion, you can repeat conversions several times, store the results in the RAM and then average them (or select the most frequently occurring values) to obtain accurate results in spite of any potential noise errors.

2.7 Register reprogramming and regular checking

It rarely happens that EMC disturbances alter the content of the registers. Generally the registers concerned are clock control registers or I/O configuration and data registers because they are close to the chip output pads.

In such cases a good security measure is to refresh these registers frequently.

Table 1. Summary of preventive techniques

Software Quality Preventive techniques	Advantage	Disadvantage	Implementation
Watchdog (Hardware or Software)	Control is CPU-independent Avoids MCU lock	Needs to be carefully handled if LP mode is used	Easy but the activation and refresh instructions must be carefully placed in the code for maximum efficiency
Force a watchdog reset in unused program memory	More direct and quicker than waiting for a watchdog timeout	Loss of previous context	Clear the WDG reset bit (see device spec.)
Fill unused program memory with software interrupt instructions	More direct and quicker than waiting for a WDG timeout.	None	Fill unused area with "TRAP" or SVC op-code and manage the failure in the corresponding interrupt routine.
A/D Converter averaging	Ensure the ADC performance in a noisy surrounding.	Processing time	Perform an iterative loop for ADC acquisitions and averaging.
Removal of illegal or critical opcode	Avoid MCU locks due to unexpected readings of WFET or WFI opcodes	None, except restriction on using these opcodes	String search in the ".list" file (see *Section 2.5*).
Input filtering	Data acquisition stability	Processing time	Repeat measurement several times and perform a statistical choice between "0" or "1".

Table 1. Summary of preventive techniques (continued)

Software Quality Preventive techniques	Advantage	Disadvantage	Implementation
Unused interrupt management	Avoid runaways due to unexpected interrupts	None	Very easy (see *Section 2.4*)
Refreshing of critical registers	Safe running	Uses MCU resources	Refresh critical registers in frequently-executed loops

2.8 Redundant data storage and exchange

All the data stored in the internal or external memory can be subject to corruption due to electromagnetic disturbance in extreme conditions.

The preventive technique of storing doubled complementary values at nonadjacent memory areas, storing and checking the parity bits or ECC are all advanced methods which help to identify and/or correct the data corruption.

Refer to AN4435 or safety manuals for more details on techniques improving software robustness available at st.com.

3 Auto-recovery techniques

This section gives some techniques for quickly recovering your application context after an EMC failure.

Unexpected resets, Program Counter jumps and parasitic interrupts are the most common EMC failures observed in the MCU whatever the source of the disturbance.

In any of these cases the RAM (or EEPROM data memory when available) remains unchanged and can be used as very efficient way to save the application context and parameters.

Note that the RAM will lose its contents if the device is powered-off. The EEPROM data keeps its content at power-off but the write time is much longer.

3.1 Saving your context in RAM

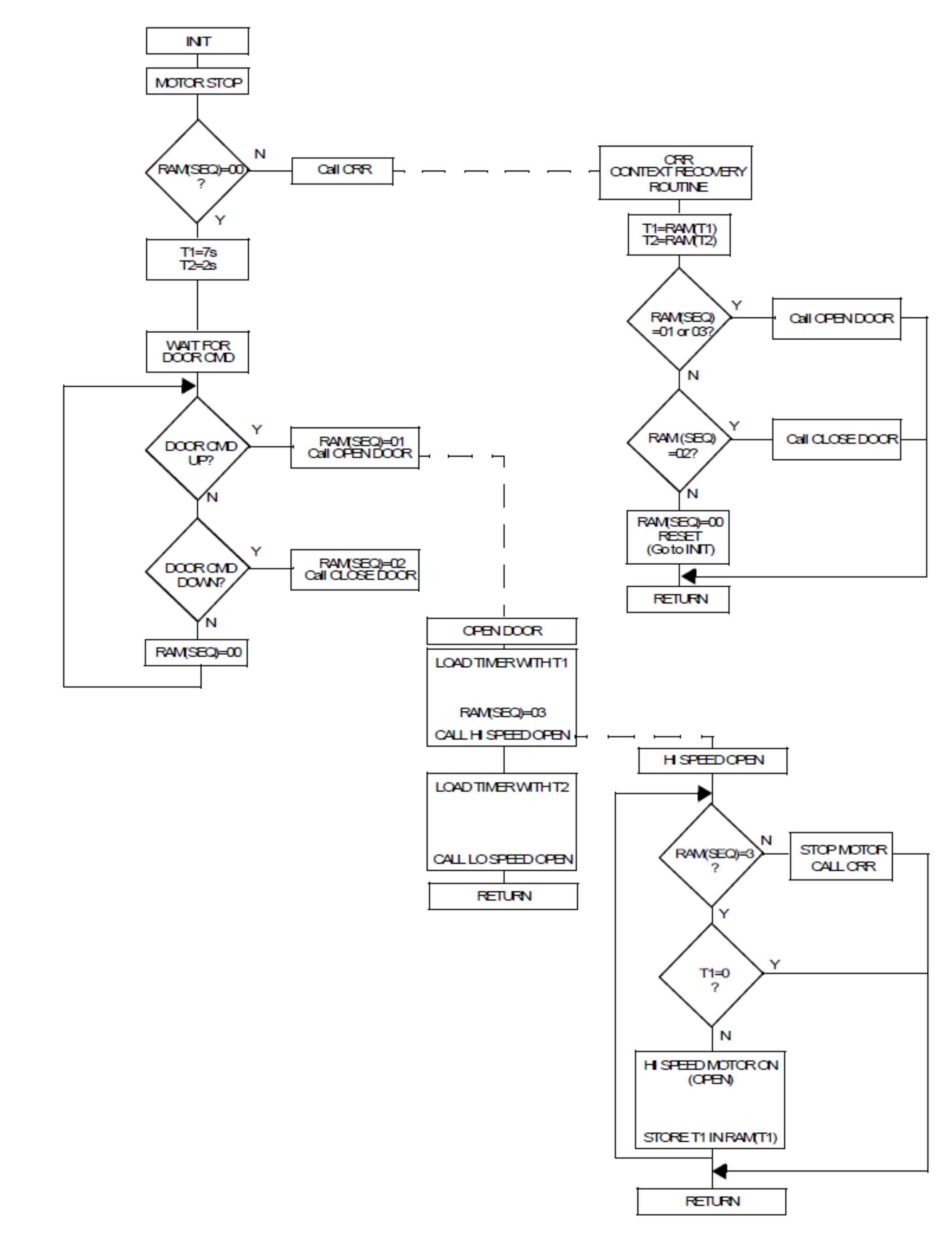
Figure 3 shows an example of a software auto-recovery implementation: the critical software sequences (door OPEN or CLOSE commands, high speed motor controls) are memorized in a RAM byte ("RAM(SEQ)").

This allows us on the one hand to recover the context if an EMC event leads to an MCU reset, and on the other hand we can check the source before a executing a critical subroutine. In this case the high speed motor activation is allowed only if RAM(SEQ)=03).

The application parameters (T1&T2 timing values) are also stored in RAM when they are changed.

This means if a software runaway event occurred or the MCU is reset (by the LVD or the watchdog), the recovery routine (CRR) will restore the last door command, reload the timing parameters and resume the program execution without any external intervention.

Figure 3. Example of auto-recovery software

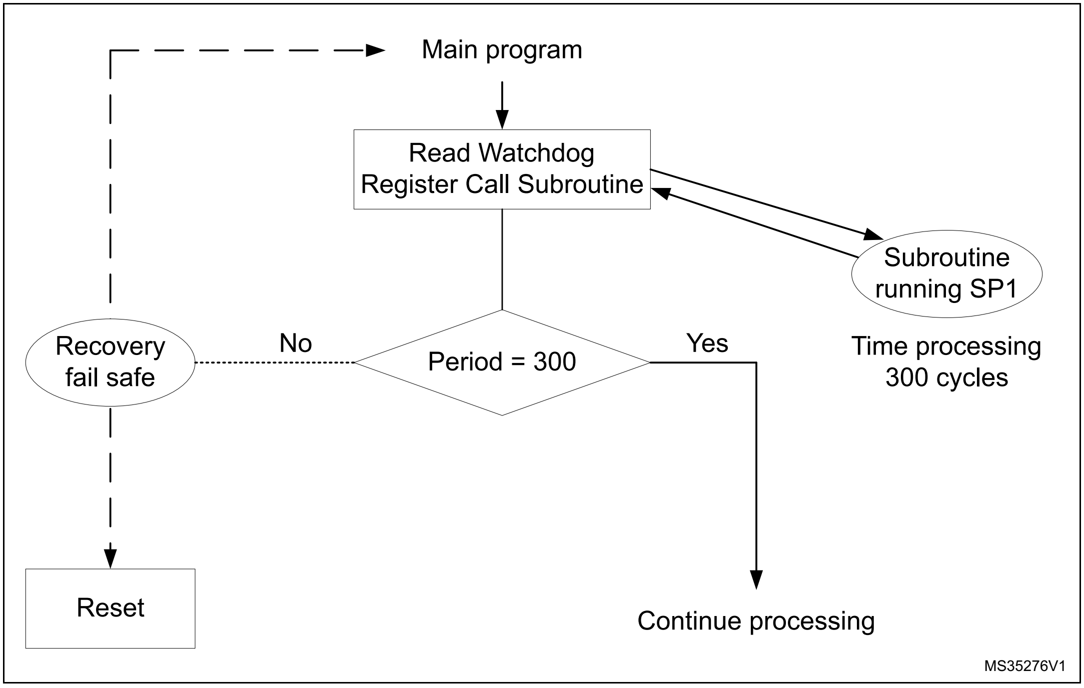


3.2 Using the watchdog for local control

Very often programmers consider the watchdog timer more or less just as a time bomb and only refresh it to its maximum value to have the widest possible margin without any direct relation to the expected program execution time.

This is a poor approach, a far better method is to use the watchdog timer register to check the execution times of individual software routines and in case of an abnormality to react promptly before the watchdog end of count and either perform an immediate reset or go into a software recovery routine.

Figure 4. Local control by the watchdog



3.3 Using the reset flags to identify the reset source

There are several possible internal reset sources: LVD (Low Voltage detector) or Watchdog reset, POR (Power On Reset), hot reset (parasitic or external reset following a low state of the Reset pin).

The reset source is flagged in a “reset register” and this information is kept as long as the MCU's power supply is on.

Figure 5 shows how you can test the reset register at the beginning of your program and then branch to a context recovery routine (depending to the detected reset source) instead of restarting the "Power On Reset" initialization routine which is often complex and time consuming.

It is very important to detect and manage parasitic resets as they are the most usual cause of microcontroller EMC failures.

Figure 5. Identify reset sources

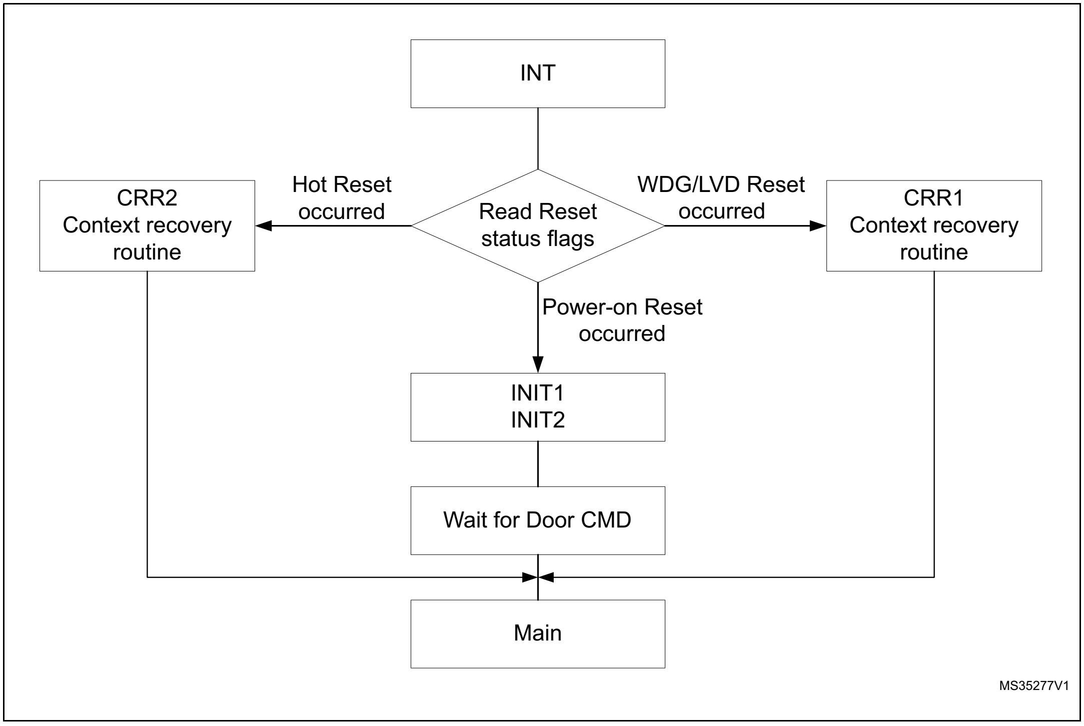


Table 2. Summary of auto-recovery techniques

Software Quality Auto-recovery techniques	Advantage	Disadvantage	Implementation
Local Control by the Watchdog	Process control of critical sequential blocks	Need to calculate an accurate time window	Check the sequence execution time using the WDG timer register
identify Reset Sources	Fast recovery from unexpected reset failures	None	Use the MCU "reset register" or the RAM to detect various reset sources.
Application context save in RAM, Flash or EEPROM	Save application parameters, ensure critical task execution resume in case of MCU failures.	Uses MCU resources	Store software critical phases and parameters in RAM, Flash or EEPROM. Use data in RAM, Flash or EEPROM to recover the last context before failure.

3.4 Saving data into non-volatile memory

The time needed to store data in the non-volatile memory (Data EEPROM) is significantly longer compared with that needed for storing data into RAM. During this programming time the system may be compromised by EMC disturbances which results in system reset terminating the programming process, resulting in the data corruption.

To prevent such situation the data should be stored in a redundant container keeping its consistency by using, among others, validation marks.

The validity of content in this data container needs to be checked after each system start before actually using it.

4 Which results can be achieved?

ST microcontrollers are designed, tested and optimized to remain fully functional with ESD voltages (according EN1000-4-2 standard) directly applied on any pin with voltage levels stated in their datasheets. Although this performance is good enough in most cases, it can be improved by using software techniques and at the same moment ensure an correct reaction of the system on EMC level which is sometimes above 4kV.

The properly designed system which is able to detect a corruption caused by EMC disturbance, initiate and successfully complete the auto recovery procedure resulting in system reset or reinitialization is always better then the system which does not detect any problem do not initiate an auto recovery mechanism and stay partially corrupted but without any visible change.

5 Revision history

Table 3. Document revision history

Date	Revision	Changes
02-Jul-2001	1	Initial release.
24-Jun-2014	2	Revised *Introduction*, *Section 2*, *Section 2.1*, *Section 2.2*, *Section 2.3*, *Section 2.4*, *Section 2.5*, *Section 3.2* and *Section 4* Added *Section 1: Related documents*, *Section 2.8: Redundant data storage and exchange* and *Section 3.4: Saving data into non-volatile memory*. Updated *Figure 3* and *Figure 4*. Updated *Table 1: Summary of preventive techniques* and *Table 2: Summary of auto-recovery techniques*.