APPLICATION NOTE

SOFTWARE TECHNIQUES FOR IMPROVING ST6 EMC PERFORMANCE

by A. Niaussat

1 INTRODUCTION

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

To achieve this goal, EMC considerations must be included as early as possible in the design phase of the project.

A quality approach to software increases the security and the reliability of the application. EMC-hardened software is inexpensive to implement, it improves the MCU's immunity performance and saves hardware costs.

Examples of software disturbances:

– Microcontroller lock

– Program Counter runaway

– Execution of unexpected instructions

– Non-execution of tests

– Bad address pointing

– Bad execution of subroutines

– Parasite reset

– Parasite interrupts

– I/O deprogramming

You can validate your EMC-hardened software in the EPROM versions which are available for all the ST microcontrollers and you can store the data in a highly secure manner using on-chip EEPROM. The microcontroller is the most sensitive part of the system because it is the system core. If electric noise affects the microcontroller's operation, the processor may enter a runaway condition and corrupt the data. All too often, programs are not designed to operate correctly when an unexpected noise spike occurs.

Examples of the consequences of failing software:

– Unexpected commands

– Loss of context

– Unexpected branch in process

– Loss of interrupts

– Loss of data integrity

– Wrong input measurement values

Two kinds of software techniques are proposed: Preventive techniques and active detection methods.

Preventive techniques are implemented during the program coding and avoid parasites and deviation from the expected process.

Active detection manages the software. When a runaway condition is detected, the idea is to take the decision to stop the microcontroller or check data integrity or give a warning or reset and return to normal operating mode. In all cases it is transparent to the user of the application.

2 EMC TRICKS

You can implement **preventive tricks** and **active detection.**

2.1 PREVENTIVE TECHNIQUES

These techniques can be implemented while writing the program, unlike active techniques (described in section 2) preventive techniques do not detect parasites.

Making good use of the Watchdog, using the average with the ADC, the stack initialization, input filtering, suppression of illegal opcodes or critical opcodes in the data space are examples of preventive software tricks.

2.1.1 Watchdog

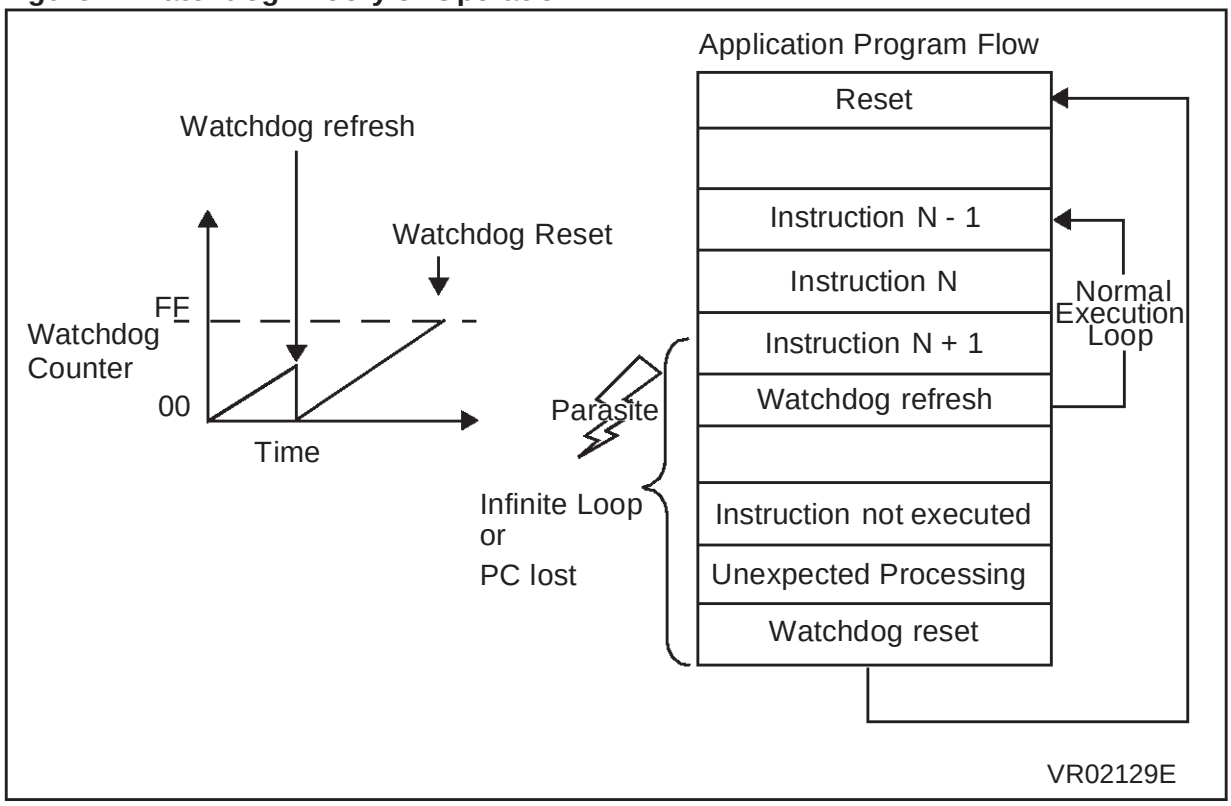
The watchdog monitors software events, it safeguards against software or hardware failure.

The watchdog must be used in the right way. The watchdog is the most common EMC resource in the MCU world. It is a parallel process that resets the microcontroller if the running software fails to refresh it in due time. The refresh periods must be in the main loop not in the subroutines or interrupt subroutines.

Simply enabling the watchdog with numerous refresh instructions inside the software is a poor way of using it. Several tricks allow the use of the watchdog feature to be considerably optimized. It is very important to minimize the period between the two refreshing periods because the MCU can do something you don’t want, an unexpected process, a runaway condition.

The use of the Watchdog resets the MCU, that means that the data space context is lost as well as the data’s integrity.

Figure 1. Watchdog Theory of Operation



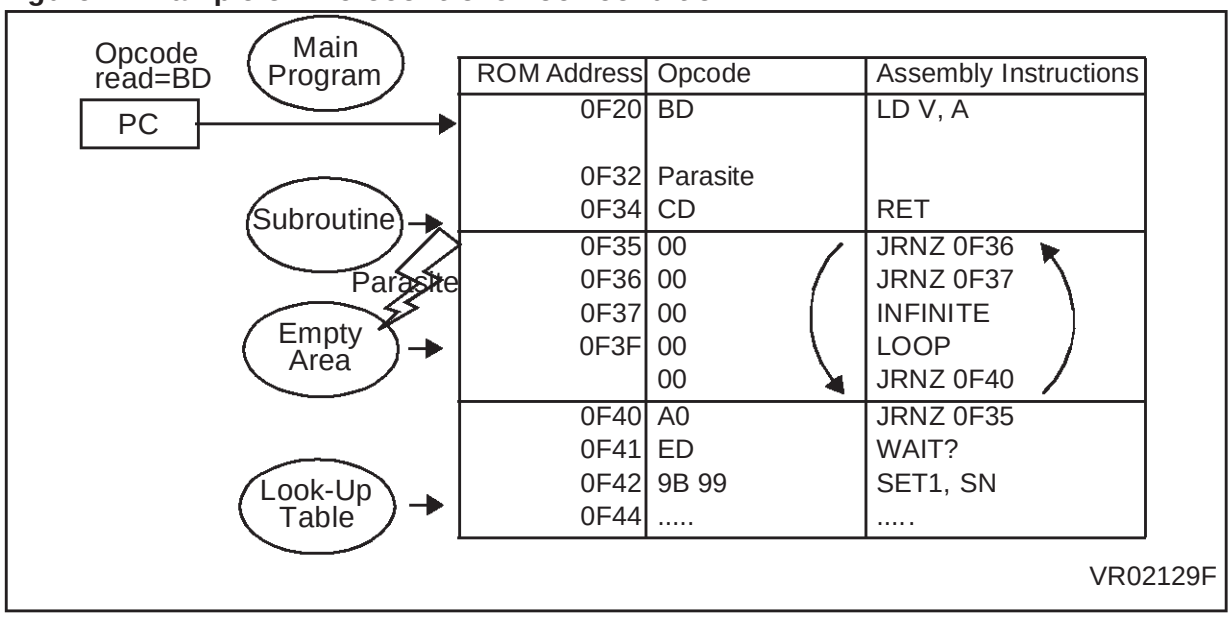
2.2 UNUSED PROGRAM AREA

In most applications, program memory space is not used completely. The corresponding memory locations must be used like this: force a Watchdog reset in the unused program memory or jump to a known place if you don't need a reset. Near the empty space in program memory, you can force a Watchdog reset if you decide that the program counter is corrupted and has no place here.

In the case below, at address 0F3F if you put **LDI WDT,01h**, this instruction causes the ST6 STMicroelectronics Microcontroller to be reset by the internal watchdog and thus avoid a microcontroller lock condition.

In this unused area you can jump to a subroutine called parasite detection subroutine, you have a parasite detector in this area.

Figure 2. Example of Microcontroller lock condition



2.3 INPUT FILTERING

This filter increases the stability of the measure, this is a simple example of the routine:

```
MAIN1  LDI LOOP,08h       repeat measure 8 times
MAIN2  JRR 4,PB,MAIN1     Check bit 4 port B
       DEC LOOP           Decrement loop
       JNRZ MAIN2         until Loop=0
```

2.4 STACK INIT

The ST6 stack has 6 levels, if a NMI parasite occurs, some data are pushed in the stack. The NMI vector branches the program to an interrupt vector which has the top priority.

Normally the PC is loaded with the address of the interrupt vector which then causes a Jump to the interrupt service routine. In case of disturbance, the PC may be loaded with a bad address. The PC continues to decode instructions and you can drop into the main program. At this moment the stack is still pushed to 1 and if others interrupts appear, they are lost.

Take care to safely initialize the stack at the beginning of the main routine like this:

```
MAIN

RETI
RETI
RETI
RETI
RETI
RETI

MAIN2
```

& other instructions

2.5 NOT USED ILLEGAL OR CRITICAL BYTE

A critical byte is an instruction like WAIT or STOP which is decoded by the microcontroller and forces it to stop executing further instruction.

In program space, when the PC is desynchronized, it may read and decode this instruction and STOP the microcontroller.

Avoid doing this: Loading the contents of a register with critical bytes ED or OD

For example, **LDI WDT,ED** because ED can be decoded like a WAIT instruction.

The illegal bytes are referenced in the MapTable; some of these bytes are interpreted by the MCU as WAIT or STOP instructions. In the ST6, these bytes are E5 & 65h. The microcontroller executes the other illegal bytes as NOP instructions.

2.6 ADC AVERAGE

The ADC average is a loop on the ADC measurement for filtering HF spikes.

Example for ST6

```
       LDI loop,04h              ; loading with nb of measurement
ADC1   LDI ADCR,30h              ; start conversion
ADC2   JRR 6,ADCR,ADC2           ; test bit 6 end of conversion
       LD A,ADR                  ; read ADC value
       ADD A,val2                ; sum
       JNRC ADC3                 ; jump relative on non carry flag
       INC val1                  ; if carry increment ADCval1
ADC3   DEC loop                  ; decrement loop variable
       JRZ ADC8                  ; end if nb of measurements = loop
       JP ADC1                   ; unconditional jump
ADC8   then divide val1 & val2 to calculate the mean
```

The preventive tricks are summarized on this table:

Software Quality Preventive Methods	Advantage	Disadvantage	Implementing
Watchdog (Hardware or Software)	Parallel Process Operational application Reset	Context lost Unexpected processing PC unsynchronized	Easy but be careful to minimize the period between the watchdog refreshes
Force a reset by the watchdog		Loss of previous context	Put one instruction in the empty program spaces LDI WDT, 01h
ADC average	Avoid parasites	Processing time for each conversion * no. of iterations	Iterative loop
No use of illegal or critical opcode	No lock (WAIT or STOP opcode not requested)	none except restriction on using these opcodes	String search with the opcode list of the MapTable
Safety stack init	In case of NMI parasite	none	6 RETI instructions at the beginning of the program
Input filtering	Measurement stability	Processing time	Repeat measurement many times

3 ACTIVE DETECTION METHODS

These methods detect parasites during the execution of program, check the state of I/O, re-program the I/O, detect a parasite reset.

An alarm is given to the software to take a decision.

You can take the decision to either:

– Stop the ST Microcontroller

– Check the data's integrity

– Recover the previous context you have previously saved

– Give an alert

– Or reset and return to the current running mode.

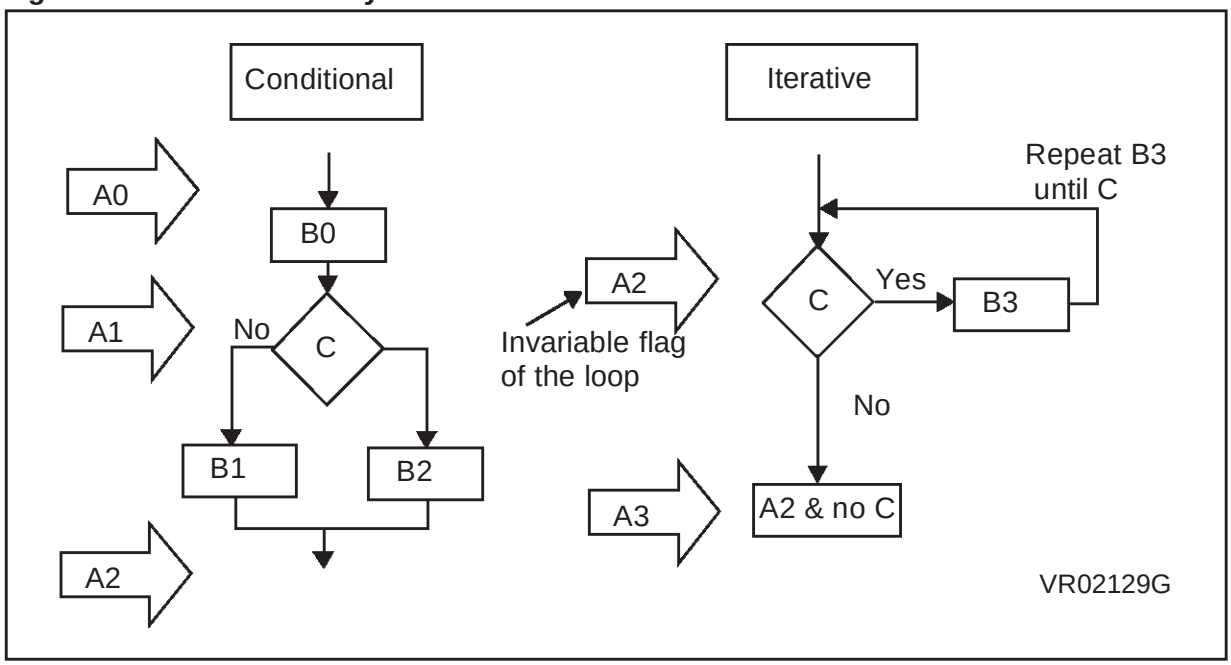
In any case it is transparent to the user of the application.

The characteristic approach of these methods is to control the flow of the program, it is based on the control structure of software validity. It is an algorithmic method.

These methods can be implemented in the control of subroutines (SDC: Subroutine Detector & Corrector) with reexecution of these subroutines. The principle is that the next task is executed only if the previous task was running correctly.

You must consider the flow as a continuation of sequential tasks that are linked by invariable flags.

Figure 3. Software validity control structure

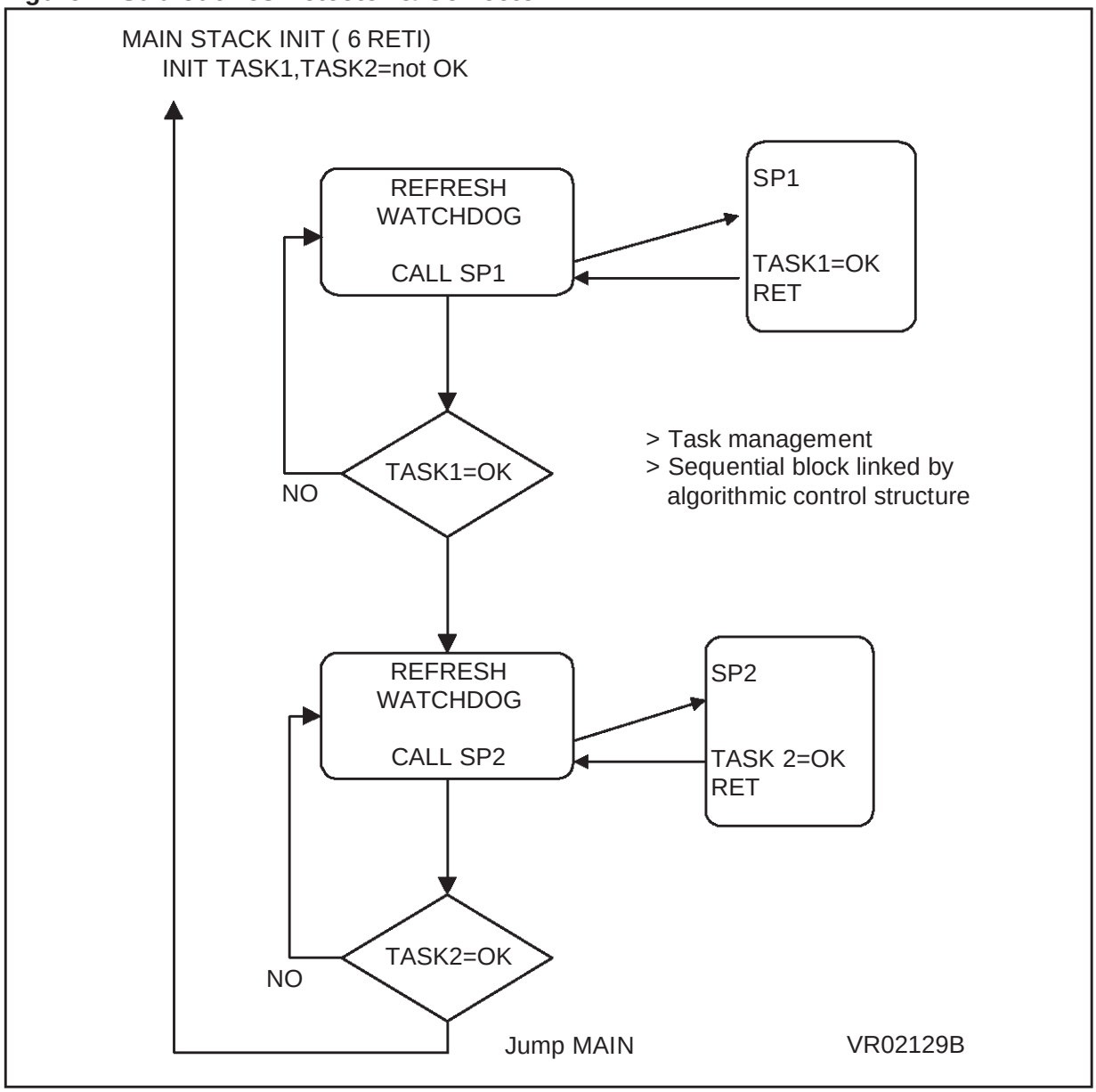


3.1 SDC SUBROUTINE DETECTOR & CORRECTOR

Invariable flags are placed in the main and in the subroutine. There is an auto control of each correctly running sequential block, if the flag control is not correct then you have detected a parasite problem.

You can branch your program on a decision module with many possible choices which depend on the application context (reexecution of subroutine, prompt reset command or control of the previous context you have already saved).

Figure 4. Subroutines Detector & Corrector



3.2 LOCAL CONTROL BY THE WATCHDOG

You can optimize the use of the watchdog by using it as a normal timer.

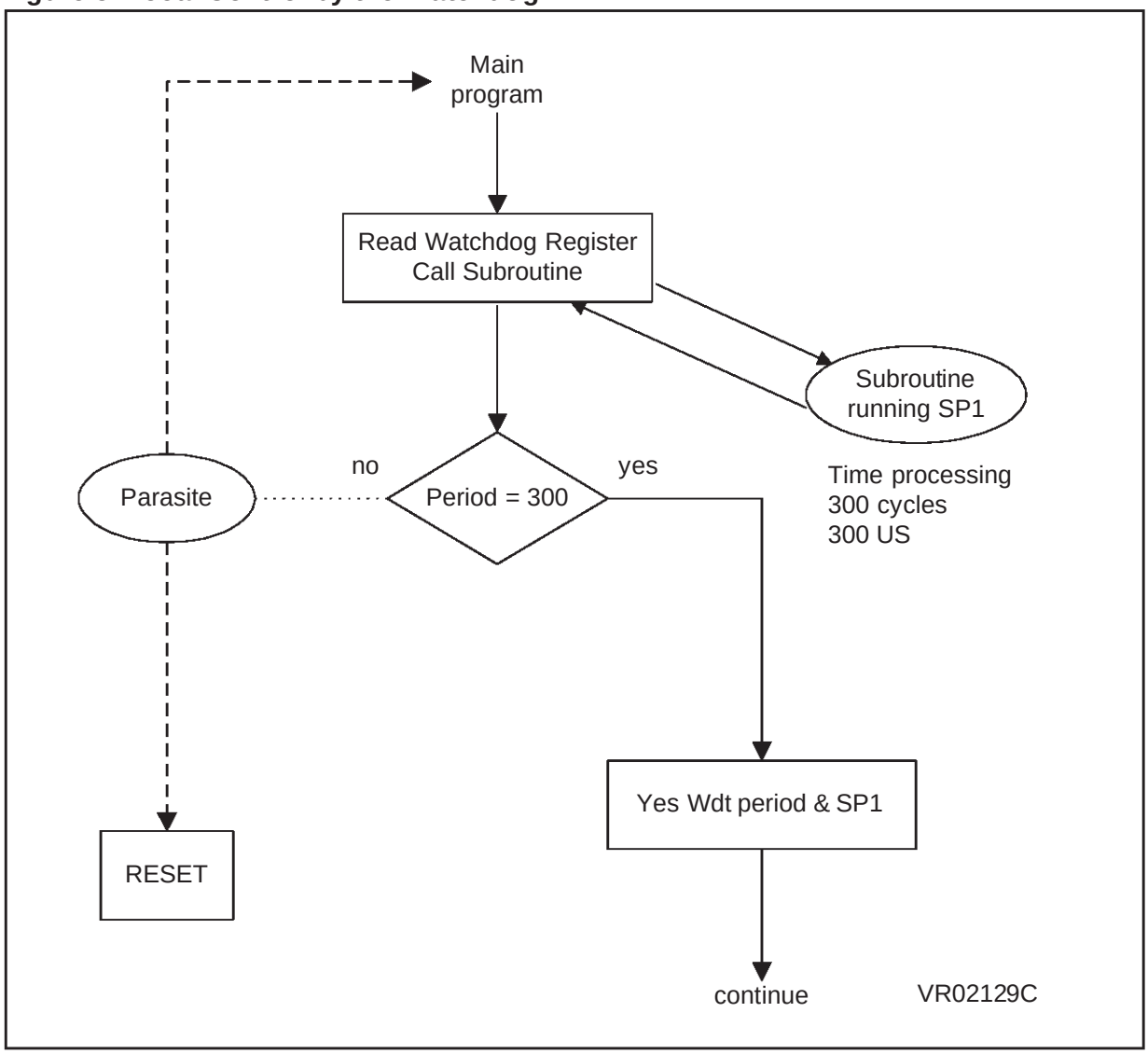
You can control the time processing of a subroutine for example by preloading the reset register and reading it after the return from interrupt.

If the time processing (number of cycles) you have calculated before is not equal to the period, you can consider that a **parasite problem has appeared.**

Decide now to reexecute the subroutine or activate the reset immediately.

This watchdog control is very precise.

Figure 5. Local Control by the Watchdog



3.3 I/O REPROGRAMMING

In case of parasite detection, verify the I/Os and reprogram them eventually.

3.4 RESET DETECTOR & CORRECTOR

There are three internal resets: LVD (Low Voltage detector), POR (Power On Reset) or Watchdog reset. There are three external resets (cold start reset when you apply the supply, reset parasite on the external reset pin reset and reset by the operator external switch)

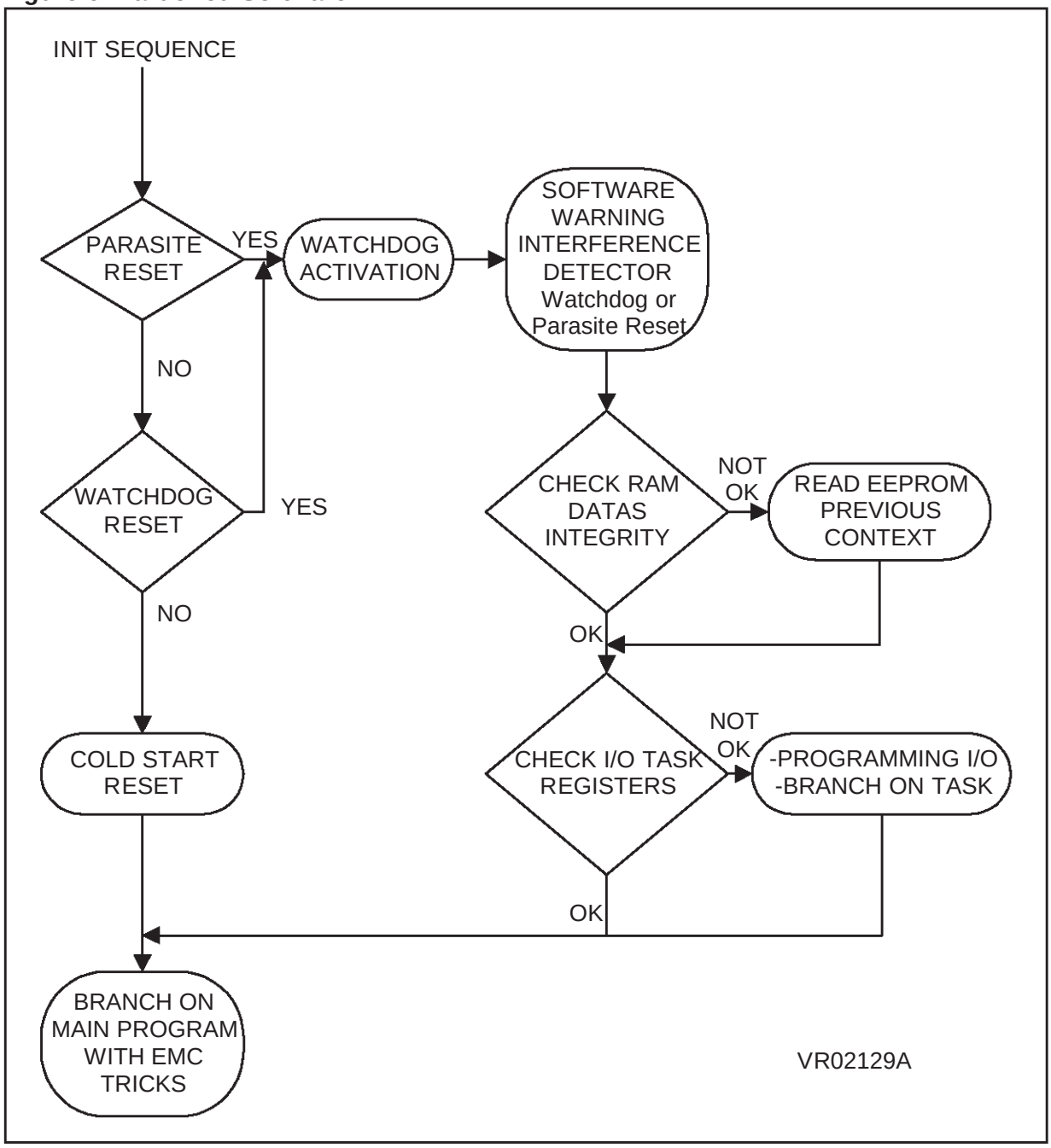
During the cold start reset, the RAM content has the same value each time you apply power to the device.

If you change this content after cold start reset by writing a value in a free address during the init, this value does not change unless you disconnect the supply. During the init phase you can check this value and distinguish the cold reset from the external (operator, parasite) or internal reset if the option byte is connected (LVD, POR, WDT).

It is possible on the ST7 & ST9 to distinguish a watchdog reset, using the reset flag register. An interrupt (in the ST9) associated with the end of watchdog count is very useful.

It is very important to detect and manage parasitic resets. For example, if the MCU was in Stop mode, a parasitic reset can wake up the MCU. The new EMC init sequence now becomes:

Figure 6. Hardened Software



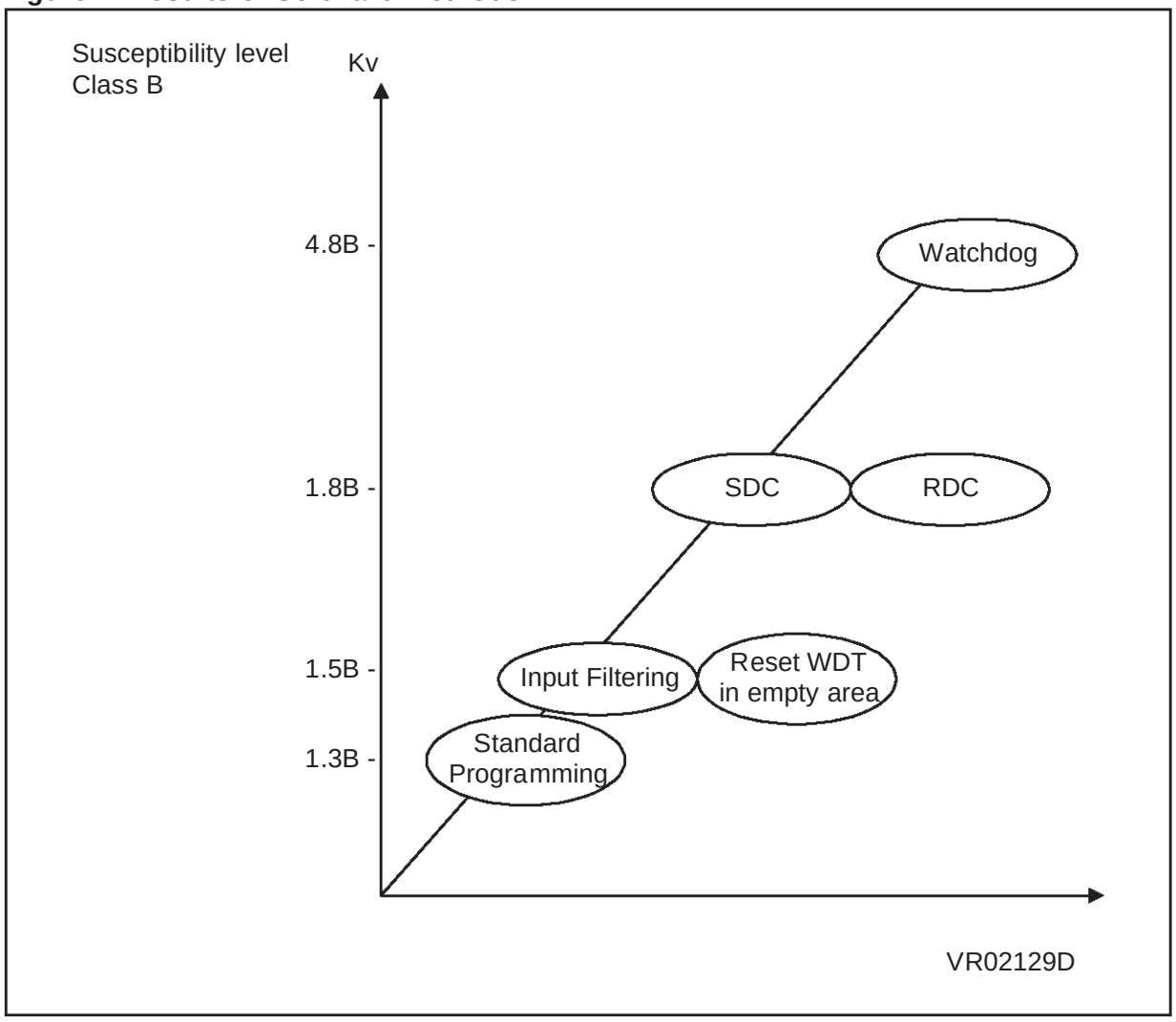
The active methods are summarized in this table.

Software Quality Preventive Methods	Advantage	Disadvantage	Implementing
Local Control by the Watchdog	Process control of each local sequential blocks	Calculation of a precise time window	Control of time window Test the Watchdog register Management of each blocks
I/O Recurring Reprogramming	Safe running	Read and test I/O	On parasite detection or automatic
Reset Detection & Correction	Parasite reset detection Context control	If software detecting a reset pressed by an operator is attributed to a parasite.	Software Priority on Reset register if exist in the architecture
Subroutines Detection & Correction	Detection & Correction Context save safety Reexecution of the flow	Tests to add	Task management Sequential block linked by algorithmic control structure Use the invariable flag of the loop

The target for the equipment is to pass the norm level 2KV Class B. That means any modification of the execution and the data content are not permitted.

After measuring each method, this is the result you will get with a standard low cost board.

Figure 7. Results of Software Methods



4 CONCLUSIONS

This application note gives programmers an idea of how to develop EMC hardened and robust software. With some additional tricks you can gain considerable improvements in EMC.

EMC must be implemented as early as possible in the design phase of the project. Bear in mind that hardening your software is a low cost way of protecting the application from potential malfunctioning.

"THE PRESENT NOTE WHICH IS FOR GUIDANCE ONLY AIMS AT PROVIDING CUSTOMERS WITH INFORMATION REGARDING THEIR PRODUCTS IN ORDER FOR THEM TO SAVE TIME. AS A RESULT, STMICROELECTRONICS SHALL NOT BE HELD LIABLE FOR ANY DIRECT, INDIRECT OR CONSEQUENTIAL DAMAGES WITH RESPECT TO ANY CLAIMS ARISING FROM THE CONTENT OF SUCH A NOTE AND/OR THE USE MADE BY CUSTOMERS OF THE INFORMATION CONTAINED HEREIN IN CONNEXION WITH THEIR PRODUCTS."