



Application Note 29

Title: **Using a small flash MCU as a fully customisable, smart supervisory circuit**

Product Family: **4-bit microcontroller, Supervisory IC**

Part Number: EM6580

Keywords: 4-bit microcontroller, supervisory, windowed watchdog, reset, voltage monitoring

Date: August 20, 2005

The EM6580 is an easy to use 4-bit flash microcontroller that includes all the features needed to be programmed as a user customisable voltage supervisory circuit that can exactly fit your applications needs. This application note details how to make a standalone supervisory circuit including windowed watchdog, pushbutton reset and supervision of a second voltage V2 along with the main supply source Vdd .

The goal of this AppNote is to provide our customers with a framework for building the supervisory circuit that is exactly right for their application.

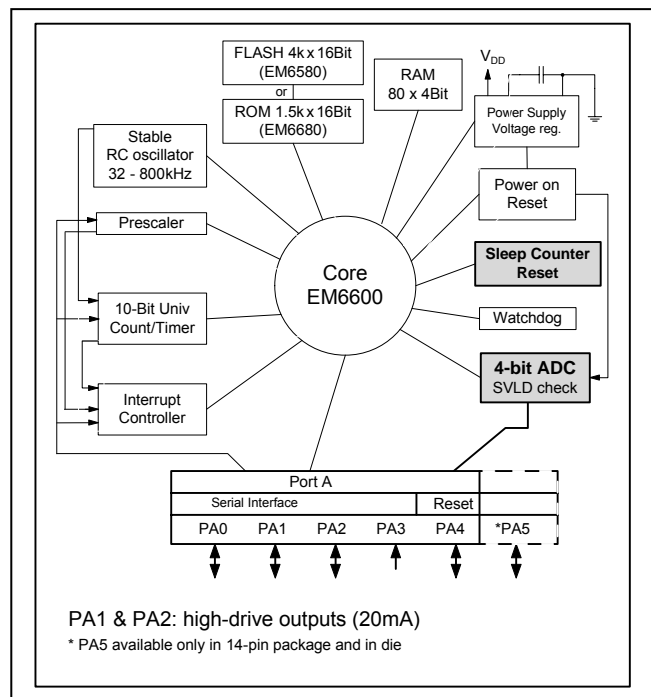
More advanced power management applications including communication by serial port to a main processor can also be implemented on this chip.

The features of the EM6580 include:

- ◆ Small 8-pin package
- ◆ True Low Current
- ◆ Flash Program Memory
- ◆ 4-bit ADC or 12 levels Voltage Level Detector
- ◆ Unique serial ID for each produced flash-device
- ◆ High drive outputs
- ◆ Max 5/6 I/Os with 2 high drive outputs of 20mA
- ◆ Power-On-Reset with brownout control
- ◆ Original EM design: Sleep Counter Reset (automatic wake-up from sleep mode)
- ◆ Internal RC oscillator 32 kHz – 800 kHz with outstanding stability
- ◆ 8-bit serial interface

Of particular interest is the true low-power capability of the EM6580 microcontroller as shown in the Table 1. EM6580 consumes as little as 5.8 µA in active mode and 0.32 µA in sleep mode.

EM6580 also contains a unique serial ID number which allows to identify each manufactured unit individually. This can be a useful feature for unit tracking, maintenance records, physical identification, etc.



		EM6580
<b>Memory</b>		Flash 4k x 16 bit
<b>Supply voltage</b>		2.0 V to 5.5 V
<b>Current</b>	active	5.8 µA
	standby	3.3 µA
	sleep	0.32 µA
<b>Serial ID #</b>		yes
<b>Package</b>		SO-8/14

Table 1

## The Stand-alone Supervisory Application

This document describes a fully programmable windowed watchdog application using the EM6580. A typical application can be seen in Figure 3 in which two supply voltage sources are monitored, the main one as Vdd and a second one V2 (battery). The application also shows a manual reset pushbutton and the application uP is monitored by the fully programmable windowed watchdog function of EM6580. Input and output polarities, voltage levels as well as all timing delays are programmable. This can be used as a basis to fulfil your proper supervisory needs.

Table 2 describes the application inputs and outputs and their programmability.

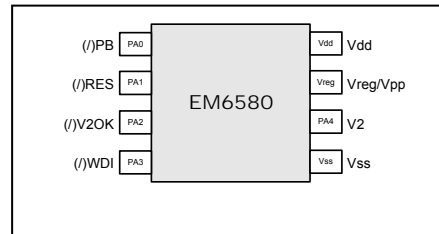


Figure 2. Application Pin-out

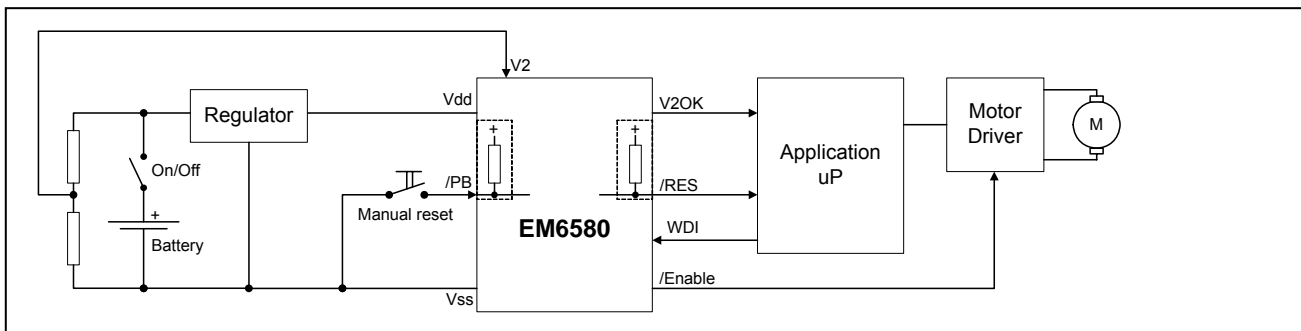


Figure 3. Application example

Input signals	Function	Pin	Internal pull-up	Flank	Debounce	Voltage
(/ )PB	Pushbutton reset input	PA0	✓	✓	✓	
(/ )WDI	watchdog clock input	PA3		✓		
V2	analog voltage 2	PA4				0V-Vdd

Output signals	Function	Pin	Active high/low	Push-pull	Open-drain	Internal pull-up
(/ )RES	Reset output	PA1	✓	✓	✓	✓
(/ )V2OK	V2 ok output	PA2	✓	✓	✓	✓
(/ )Enable	increased confidence enable	PA5	✓	✓	✓	✓

Table 2

### Voltage monitoring

The EM6580 has a single voltage comparator the input of which can be multiplexed between the pin PA4 and the Vdd pin. For voltages that only change slowly like battery voltages, however, discontinuous monitoring can be useful and acceptable. In this implementation Vdd will be monitored quasi-continuously, the V2 input intermittently.

### 9th Pin

The EM6580 is available in SO-8 and SO-14 packages and in die form. The SO-14 and die chips have a 9<sup>th</sup> pin connected, PA5. This is used in this application for an Enable output. This pin is not available, however, in the 8-pin package.

## Watchdog timing

This program implements a windowed watchdog. The essential features of a windowed watchdog are that a WDI input signal being received during the closed window causes a reset, as does the passing of the entire open window without a WDI input signal. This implementation includes an additional timeout, the first cycle timeout, after each reset pulse to allow more time for initialising the external application before starting the normal program execution. There are two power saving modes. In PowerSaveState the time between reset pulses is increased after 5 reset pulses elucidate no response from the external application. In BadVddState the watchdog monitors the re-establishment of the power supply voltage approx 4 times a second. In between the MCU is placed in Sleep Mode which ensures the absolute minimum power consumption.

The signal behaviour can be seen in the Figure 4. At (1) Vdd passes the threshold and /RES is released. This starts Tfc followed by Tcw. During this time no WDI input is allowed. During Tow, however, a WDI input is required. The next Tcw starts immediately with the active edge of WDI. The enable output is asserted after 3 good watchdog cycles (2) as an enhanced confidence signal. It can be used, for example, to gate motor drivers.

At (3) Tow runs out without a WDI input, this causes an immediate reset and the removal of Enable. At (4) a WDI input during Tcw causes /RES to be asserted. Note that Tfc is inserted after each /RES assertion whereas restarting the cycle with WDI starts immediately at Tcw. At (6) we see the change in behaviour to save power after 5 non-successful resets. At (5) and (7) we see that a watchdog cycle that is started by a reset output commences with Tfc, while a cycle started by a WDI input commences immediately with Tcw.

The windowed watchdog parameters as implemented here are:

First cycle timeout	Tfc
Open window	Tow
Closed window	Tcw
Reset pulse length	Trp
Power-save mode t/o	Tps

Note that these times, as well as the entire watchdog structure are freely programmable by software to suit your application.

Note that this is an example application and all the signal levels and active flanks as well as all other parameters can be simply changed as needed for your application.

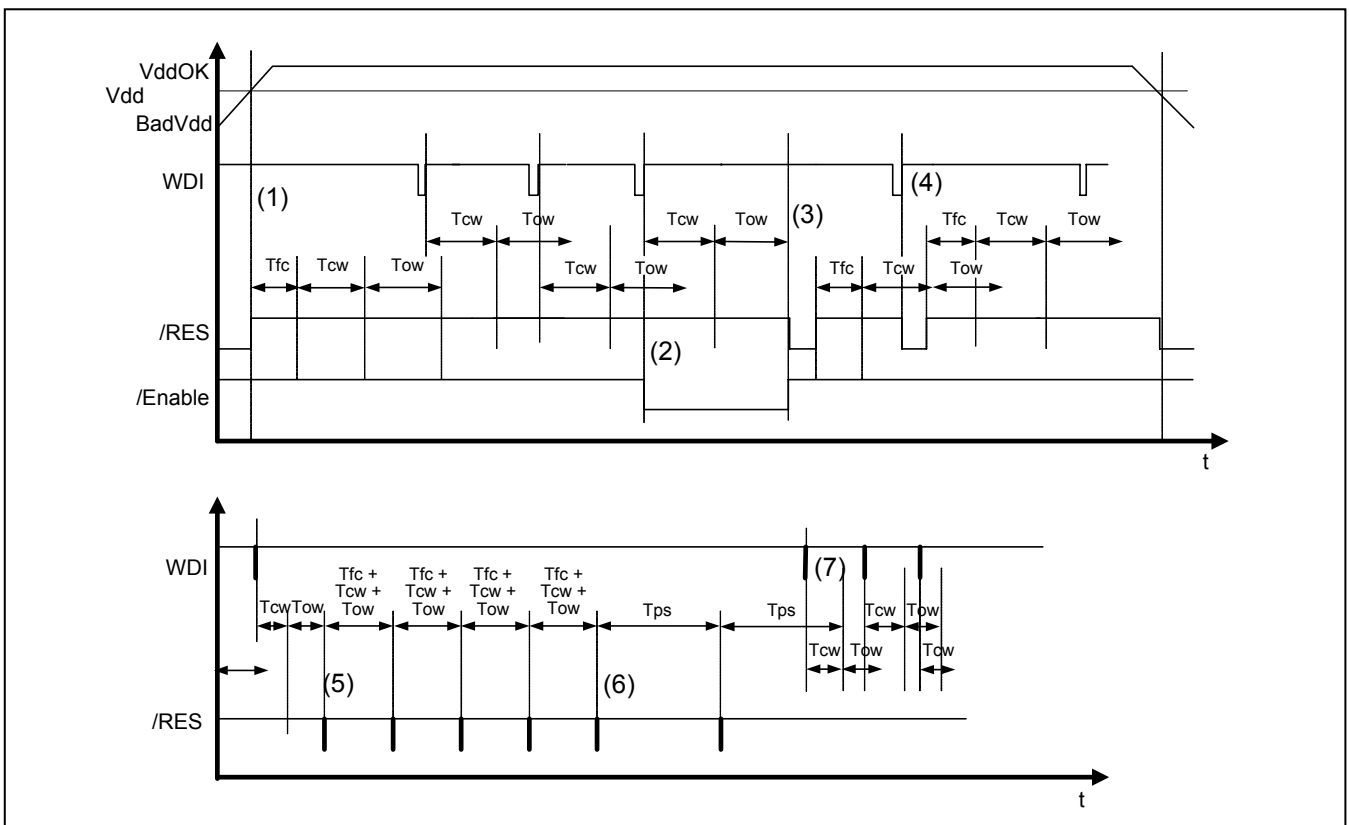


Figure 4. Watchdog timing



## Software structure

To reduce the circuit power consumption to the minimum, the software for this supervisory circuit is interrupt-driven as much as possible. Most actions are done by placing the circuit in a state and then going into Standby mode to wait for an interrupt, which indicates a state change. The program uses only one level of stack, for the interrupt routine. The interrupt routine is used only to determine the source of the interrupt, which is passed to the main program as a trigger for state changes.

The functions implemented are:

- ◆ Pushbutton reset input
- ◆ Monitoring of minimum Vdd after POR before releasing Reset
- ◆ Windowed watchdog with prolonged power-on-reset POR cycle
- ◆ Reset pulse output
- ◆ Enable output after 3 consecutive good watchdog cycles
- ◆ Automatic power-save mode after 5 consecutive bad watchdog cycles
- ◆ Use of sleep mode to reduce the power consumption to the absolute minimum when Vdd is too low
- ◆ Monitoring of a second analog input voltage (low speed)

The program state diagram is in Figure 5:

## SW Parameters

First cycle timeout	TfcrCnt
Open window	TowCnt
Closed window	TcwCnt
Reset output pulse length	TrpCnt
PowerSave timeout	TpsCnt
V2 input OK level	V2Oklevel
Minimum Vdd level	BadVddLevel

Being able to program these values individually makes it possible to customise this part very exactly to your needs, which is often difficult using standard chips.

## Reducing Power Consumption

Several actions are taken to reduce the power consumption of the chip. As mentioned above the program is mainly interrupt-driven allowing the processor to spend most of its time in Standby state, not executing, but with all the peripherals running. This reduces the power consumption by more than 30% typically.

In a first step to use even less power the frequency of the reset pulses is reduced if more than 5 consecutive bad watchdog cycles have been executed. In this case the program is in PowerSaveState and the time between reset pulses is determined by the parameter TpsCnt instead of TcwCnt+TowCnt. By making TpsCnt significantly larger than TcwCnt+TowCnt the processor

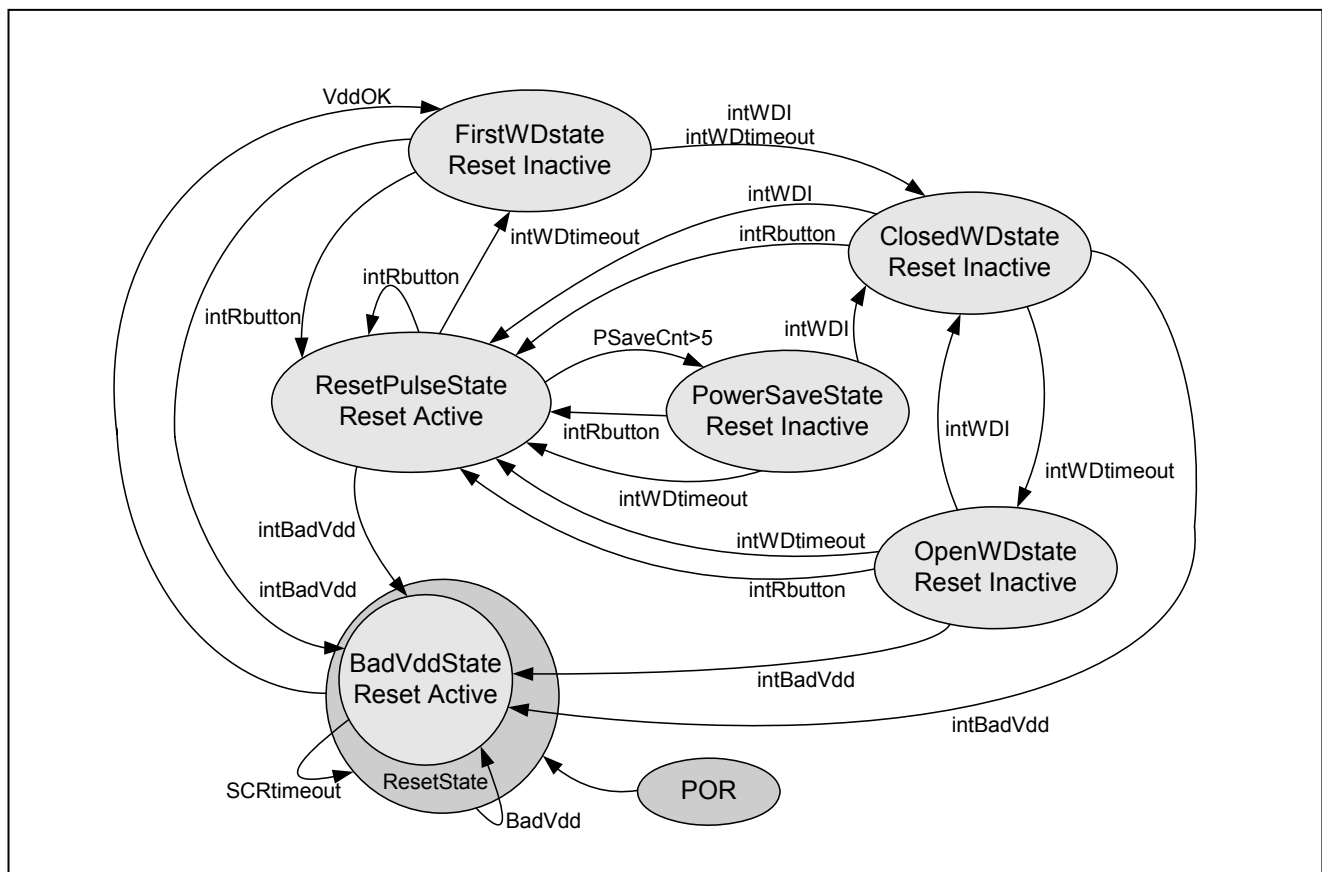


Figure 5. State diagram



will wake-up less often and spend a larger percentage of its time in Standby.

When the battery voltage is below the minimum level as defined in BadVddLevel the program switches to strict power conservation. In BadVddState we no longer need to maintain continuous battery voltage surveillance, but need to recognize when the power supply is up again to initiate a restart. At the same time the weak batteries must be spared as much as possible. To this end the program goes into sleep mode when in BadVddState. The Sleep Mode Counter is set to wake the processor

periodically to allow checking if Vdd has risen to a good level. In sleep mode both the processor core and the peripherals are turned off except for the Sleep Mode Counter and the Wake-up on Change input function. The power reduction in Sleep mode is an additional factor 10 from the Standby power consumption. The time between wakeup resets is chosen to be less than the EM6580 internal watchdog timeout, preventing the MCU from going into a hardware watchdog reset.

### Actions at State Transitions

State	On Entering	On Exiting
FirstWDstate	Set Reset inactive Set Enable inactive EnableCnt = 0 Test V2 and set V2OK accordingly Start continuous Vdd check Load counter with TporCnt Enable interrupts Halt	intWDtimeout → ClosedWDstate intWDI → ClosedWDstate intRbutton → ResetPulseState intBadVdd → BadVddState
ClosedWDstate	Set Reset inactive If EnableCnt >= 3 set Enable active else set Enable inactive Test V2 and set V2OK accordingly Restart continuous Vdd check Load counter with TcwCnt Enable interrupts Halt	intWDI → ResetPulseState intWDtimeout → OpenWDstate intRbutton → ResetPulseState intBadVdd → BadVddState
OpenWDstate	Set Reset inactive If EnableCnt >= 3 set Enable active else (set Enable inactive, EnableCnt++) Load counter with TowCnt Enable interrupts Halt	intWDI → PsaveCnt=0; ClosedWDstate intWDtimeout → ResetPulseState intRbutton → ResetPulseState intBadVdd → BadVddState
ResetPulseState	Set Enable inactive Set Reset active EnableCnt = 0 if PsaveCnt <=5 PsaveCnt ++ If Rbutton pressed loop Load counter with TrpCnt Enable interrupts Halt	intWDtimeout & SleepCnt>5 → PowerSaveState intWDtimeout & SleepCnt<=5 → FirstWDstate intRbutton → ResetPulseState intBadVdd → BadVddState
PowerSaveState	Set Enable inactive Set Reset inactive EnableCnt = 0 Test V2 and set V2OK accordingly Restart continuous Vdd check Load counter with TpsCnt Enable interrupts Halt	intWDtimeout → ResetPulseState intRbutton → ResetPulseState intWDI → ClosedWDstate intBadVdd → BadVddState
BadVddState	Set Enable inactive Set Reset active EnableCnt = 0 PsaveCnt = 0 Check Vdd if VddOK → FirstWDstate Load SCR counter Sleep	VddOK → FirstWDstate SCRtimeout → ResetState
ResetState	Initialise peripherals	→ BadVddState



## Hardware configuration

### Assigning I/O Pins

- The EM6580 ADC external input is on PA4
- The two input debouncing units are assignable to PA0 or PA5 and PA3 or PA4
- The two input wakeup on change units are assignable to PA0 or PA5 and PA3 or PA4
- All pins have programmable pull-up/pull-down
- All pins can have the up drive removed to emulate open drain

### Programming the inputs

#### Pushbutton Reset, PA0

- Functions:
- Input → RegPa0OE, bit OEnPA[0] = 0
  - Active Flank → RegPACnt1, bit EdgeFallingPA[0/5] = 0, for rising edge, =1 for falling edge
  - Debounce → RegPACnt1, bit DebounceNoPA[0/5] = 0
  - Wake on change → RegPACnt2, bit WUchEnPA[0/5] = 1
  - IRQ → RegPACnt2, bit IrqPA[0/5] = 0
  - Pull-Up → Pa0noPDown, bit NoPdPA[0] = 1
  - Pa0NchOpenDr, bit NchOpDrPA[0] = 1

#### WatchDogInput WDI, PA3

- Functions:
- Input → RegPa0OE, bit OEnPA[3] = 0
  - Active Flank → RegPACnt1, bit EdgeFallingPA[3/4] = 0, for rising edge, =1 for falling edge
  - No debounce → RegPACnt1, bit DebounceNoPA[3/4] = 1
  - Wake on change → RegPACnt2, bit WUchEnPA[3/4] = 1
  - IRQ → RegPACnt2, bit IrqPA[3/4] = 0

#### Voltage 2, V2, PA4

- RegSleepCR, bit NoPullPA[4] = 1, removes pull-up from the input
- RegVIdCntl, bit extVcheck = 1, during the time the external voltage V2 is being tested

### Programming the outputs

#### Reset Output, PA1

- Functions:
- Output → RegPa0OE, bit OEnPA[1] = 1
  - Open Drain → Pa0noPDown, bit NoPdPA[1] = 1
  - Pa0NchOpenDr, bit NchOpDrPA[1] = 1
  - or
  - Push/Pull → Pa0noPDown, bit NoPdPA[1] = 1
  - Pa0NchOpenDr, bit NchOpDrPA[1] = 0

#### Voltage 2 OK, V2OK, PA2

- Functions:
- Output → RegPa0OE, bit OEnPA[2] = 1
  - Open Drain → Pa0noPDown, bit NoPdPA[2] = 1
  - Pa0NchOpenDr, bit NchOpDrPA[2] = 1
  - or
  - Push/Pull → Pa0noPDown, bit NoPdPA[2] = 1
  - Pa0NchOpenDr, bit NchOpDrPA[2] = 0

#### Enable, E, PA5

- Functions:
- Output → RegPA1, bit OEnPA[5] = 1
  - Open Drain → RegPA1, bit NchOpDrPA[5] = 1
  - RegPACnt4, bit NoPdPA[5] = 1
  - or
  - Push/Pull → RegPA1, bit NchOpDrPA[5] = 0
  - RegPACnt4, bit NoPdPA[5] = 1



## Programming the Timer/Counter

The EM6580 10-bit counter will be used to provide all the timeout functions of the supervisor using interrupts. This allows placing the MCU in standby mode to reduce the power consumption. Typical windowed watchdogs allow fixing the Twd between 10ms and 1.5s. The reset pulse that is output is around 2.5ms (Trp) in general. Running the EM6580s 10-bit counter with the prescaler clock at 2kHz gives us a resolution of 0.5ms and a maximum timeout of 511ms. For longer intervals the 512Hz prescaler clock can be used with results in a 2ms resolution and a maximum timeout of 2047ms.

It must be taken into account that the longest timeout must be shorter than the timeout of the EM6580 internal watchdog function so that this is never activated during normal functioning.

For this example the following specification will be used:

POR timeout	Tfc	10ms
Open window	Tow	40ms
Closed window	Tcw	30ms
Reset output pulse	Trp	2.5ms
PowerSave timeout	Tps	0.5s

It is also assumed that a standard EM6580 part will be used with the trimmed 32kHz clock.

For this application we will choose to use the prescaler clock Ck[12] which is 2kHz as the counter clock source.

## Programming the prescaler

Prescaler clock 2kHz, downcounting

→RegCCnt1 = 0x2

10-bit compare

→RegCDataH, bit BitSel[1&0] = 0

The counter values for the different timeouts are simply calculated from the desired timeout length and the bit count time of 0.5ms:

POR timeout	Tfc	10ms	→ 20
Open window	Tow	20ms	→ 80
Closed window	Tcw	15ms	→ 60
Reset output pulse	Trp	2.5ms	→ 5
PowerSave cycle time	Tps	0.5s	→ 1000

To start a timeout first stop the counter

RegCCnt2 bit Start = 0,

then load the counter value through the registers RegCDataL, RegCDataM and RegCDataH and load it in the counter itself by writing

RegCCnt2 bit Load = 1.

The counter is started by setting

RegCCnt2 bit Start = 1.

To enable an interrupt when the counter has reached zero write

RegIRQMask1 bit MaskIRQCount0 = 1

and enable interrupts in general by setting

IntEn = 1 in RegSysCnt1.

## Programming the Sleep Counter

To reduce the chip power consumption to an absolute minimum in BadVccState the processor is placed in sleep mode. Periodically it is woken to check if the power supply voltage has risen up to a valid level again. At the same time it must be ensured that the processor watchdog does not provoke a hardware reset. To this end we can either turn off the watchdog during this time or program the sleep counter for a shorter timeout than the watchdog timer. We choose to leave the processor watchdog running at all times in this application and program the sleep counter accordingly with a nominal 237ms timeout.

RegSleepCR = x001B

We must, of course, not forget to reset the watchdog periodically, at least every 2.5 seconds, by setting the bit WDReset in the register RegSysCnt2. Note that after any reset the watchdog is automatically enabled.

## Programming the ADC for supply voltage level monitoring

This supervisory chip monitors 2 voltages, the chip power supply voltage Vdd and the V2 input pin. It is assumed that the V2 voltage changes only slowly, like a battery voltage, so that continuous monitoring is not necessary. As the EM6580 has one ADC unit, the Vdd voltage is not monitored during the short periods when the V2 voltage is checked. In the present implementation the V2 is monitored once during each ClosedWDstate, which will give it a maximum reaction time of about 140ms with the timings used in this AppNote. V2 is also monitored in PowerSaveState and BadVddState so that coverage is not lost at any time, but the monitoring interval will be longer. The V2OK output is asserted when the input voltage on the pin V2 is higher than V2OKlevel and negated when the voltage is lower. In this implementation the state of V2 is not coupled to the reset output.

During all the time when V2 is not actively being tested the SVLD is testing the Vdd input continuously, comparing it to the voltage level defined by BadVddLevel. The SVLD conversion time is 260µs typical, 382µs max using a 32kHz processor clock. The interrupt caused by the Vdd input falling below the BadVddLevel can arrive and must be accepted at any point during the execution of the program.

As the input voltage must never be more than Vdd+0.3V a resistive divider will often be necessary. When doing the calculations for this divider, be sure to take into consideration the input leakage current, which is small, but non-negligible. The specified typical input impedance is 900MΩ.



## Programming the SVLD and ADC voltage levels

The active voltage level is set in the register RegSVLDlev. The register values and the corresponding voltage are:

RegSVLDlev	SVLD voltage	ADC voltage
0	-	0.50V
1	-	0.65V
2	-	0.80V
3	-	0.95V
4	-	1.10V
5	-	1.25V
6	-	1.40V
7	-	1.55V
8	-	1.70V
9	-	1.85V
10	2.00V	2.00V
11	2.15V	2.15V
12	2.30V	2.30V
13	2.45V	2.45V
14	2.60V	2.60V
15	2.75V	2.75V

To monitor the Vdd voltage the SVLD input must be from the Vdd pin and not PA4

RegVldCntl, bit ExtVCheck =0.

For continuous monitoring the SVLDen bit must be set after determining the desired SVLD voltage using the RegSVLDlev register. Setting MaskIRQSVld to 1 and enabling interrupts will cause an interrupt whenever the voltage level descends below the determined level. For continuous monitoring this only needs to be started once. It must be noted however that during the voltage measurement the chip power consumption increases by about 4µA. During continuous monitoring this 4µA is drawn continuously.

To switch over to measuring the voltage on the pin PA4, in our case V2, it is only necessary to disable the interrupt for safety, stop the continuous comparison, SVLDen = 0, change the input by setting extVcheck = 1, setting the required voltage level in SVLDlev and starting a measurement with VLDstart. The measurement is finished when the flag VLDBusy returns to 0. The bit VLDRresult is set to 0 if V2 is lower than SVLDlev and set to 1 if it is higher. The VLDRresult bit is not valid as long as VLDBusy is 1. An interrupt can be generated if the voltage is lower than SVLDlev.

One should be sure to disable the SVLD interrupt before changing any of the registers concerning the SVLD.

## Interrupts

The EM6580 has 8 interrupt sources. Each interrupt source can be enabled separately by setting the corresponding bit in the registers RegIRQMask1 or RegIRQMask2 to a 1. The general interrupt enable is by the IntEn bit in the register RegSysCntl1. Entering Standby mode also automatically enable interrupts. The state of the interrupt sources can be read in the registers RegIRQ1 and RegIRQ2. Reading these registers automatically resets them so it is necessary to save a copy of the register after reading it. Interrupts are only

recognized when a positive flank from the peripheral interrupt output is seen while the interrupt mask bit is set.

This is a very simple, but efficient interrupt structure, which leaves all the processing and vectorization to the program. In this example the interrupt routine does no more than save the interrupt source and return to the position where the processor went into standby. Processing continues according to the interrupt that was received and the state in which it was in at that time.

As long as the corresponding bit in the RegIRQMask1 or 2 register is set, an incoming interrupt is saved, even if the general interrupt enable is not set. The saved interrupt will be seen immediately when interrupts are enabled, either by explicitly setting IntEn in RegSysCntl1 or by executing the HALT instruction, which places the processor in standby and enables the interrupts.

## Timer Calibration

During the production test phase the correct values for calibrating the internal RC clock are programmed in the flash memory. To enjoy the benefits of this calibration, however, they must be written into the RegOscTrim registers. A routine to do this is also in the flash memory, it is only necessary to call the routine Trim\_osc\_base\_512kHz which is at the address 0FE0H after each POR.

When calculating the timeout periods as required for your application be sure to factor in the clock tolerances. The factory calibration values assure tolerances of +5% at ambient temperature and +-10% over the full -20°C to +85°C temperature range

## Watchdog Service

As the EM6580 itself is a microcontroller it is prudent to use its own internal watchdog to be sure that it is working correctly itself. This internal watchdog is not to be confused with the watchdog implemented in this application.

To assure that the chip watchdog is monitoring the application running it is essential that the watchdog reset commands be made from inside the application. A separate interrupt routine that does nothing but service the watchdog is useless as it is outside the control logic of the application. In this application the watchdog is serviced at each state transition.

## Other Applications

For applications needing more output pins for on/off control or other functions other versions of the EM65XX/EM66XX family, such as the EM6607, provide the same ultra-low power functionality in larger packages with more pins. EM6680 is the ROM-version of EM6580 and can be used as an alternative for high volume production.



## APPENDIX. Software Program

```
-----  
;                                     SUPERVISORY.ASM  
;  
; EM6580 supervisory application  
; D.W.Corson, EM Microelectronic  
; 12.7.2005  
; Rev. F, 11.8.2005  
; Disclaimer:  
; This program has been tested and is believed to work as described. It has,  
; however, not undergone full industrial testing. Be sure to test completely  
; for your application.  
;  
; EM Microelectronic-Marin SA (EM) makes no warranty for the use of its  
; products, other than those expressly contained in the Company's standard  
; warranty which is detailed in EM's General Terms of Sale located on the  
; Company's web site. EM assumes no responsibility for any errors which  
; may appear in this document, reserves the right to change devices or  
; specifications detailed herein at any time without notice, and does not  
; make any commitment to update the information contained herein. No  
; licenses to patents or other intellectual property of EM are granted in  
; connection with the sale of EM products, expressly or by implications.  
; EM's products are not authorized for use as components in life support  
; devices or systems.  
;  
-----  
; 6580 registers  
; INCLUDE V6680REG.ASM  
;  
; Constants  
-----  
Trim_osc_base_512kHz EQU 0FE0H ; calibrate 32kHz call address  
; ----- SVLD levels  
SVLD050 EQU 0 ; 0.50Volt  
SVLD065 EQU 1 ; 0.65Volt  
SVLD080 EQU 2 ; 0.80Volt ... etc.  
SVLD095 EQU 3  
SVLD110 EQU 4  
SVLD125 EQU 5  
SVLD140 EQU 6  
SVLD155 EQU 7  
SVLD170 EQU 8  
SVLD185 EQU 9  
SVLD200 EQU 10  
SVLD215 EQU 11  
SVLD230 EQU 12  
SVLD245 EQU 13  
SVLD260 EQU 14  
SVLD275 EQU 15  
BadVddLevel EQU SVLD230 ; level below which causes reset  
V20KLevel EQU SVLD155 ; level below which causes V20K inactive  
GoodEnableCnt EQU 3 ; nr. of good WD cycles until enable goes active  
EnterPowerSaveCnt EQU 5 ; nr. of bad WD cycles until power save starts  
  
; ----- counts calculated for 2kHz counter clock  
;  
TfcCnt EQU 20 ; 10ms extra for first cycle after reset  
TowCnt EQU 80 ; 40ms open window  
TcwCnt EQU 60 ; 30ms closed window  
TrpCnt EQU 5 ; 5ms length of reset pulse  
TpsCnt EQU 1000 ; 0.5s cycle time in power save mode  
  
-----  
; Variables  
-----  
  
STACK0 EQU 3FH ; stack variable to save ACCU during interrupt  
SINT1 EQU 40H ; save interrupt state 1  
SINT2 EQU 41H ; save interrupt state 2  
EnableCnt EQU 42H ; count good watchdog cycles  
PSaveCnt EQU 43H ; count bad watchdog cycles
```



```
-----  
; Macros  
-----  
;  
; Macros for reset, enable & v2ok outputs, active positive or negative  
  
assert_reset macro ; reset on PA0 bit 1, pin 2  
; -- hi active output: for hi active output uncomment this portion  
; LDI 02H  
; OR RegPA0  
; -- end hi active output  
; -- low active output: for low active output uncomment this portion  
; LDI NOT 02H  
; AND RegPA0  
; -- end low active output  
; STA RegPA0  
; macend  
  
clear_reset macro  
; -- hi active output: for hi active output uncomment this portion  
; LDI NOT 02H  
; AND RegPA0  
; -- end hi active output  
; -- low active output: for low active output uncomment this portion  
; LDI 02H  
; OR RegPA0  
; -- end low active output  
; STA RegPA0  
; macend  
  
; -----  
assert_enable macro ; enable on PA1 bit 1, pin 9  
; -- hi active output: for hi active output uncomment this portion  
; LDI 02H  
; OR RegPA1  
; -- end hi active output  
; -- low active output: for low active output uncomment this portion  
; LDI NOT 02H  
; AND RegPA1  
; -- end low active output  
; STA RegPA1  
; macend  
  
clear_enable macro  
; -- hi active output: for hi active output uncomment this portion  
; LDI NOT 02H  
; AND RegPA1  
; -- end hi active output  
; -- low active output: for low active output uncomment this portion  
; LDI 02H  
; OR RegPA1  
; -- end low active output  
; STA RegPA1  
; macend  
  
; -----  
assert_v2ok macro ; v2ok on PA0 bit 2, pin 3  
; -- hi active output: for hi active output uncomment this portion  
; LDI 04H  
; OR RegPA0  
; -- end hi active output  
; -- low active output: for low active output uncomment this portion  
; LDI NOT 04H  
; AND RegPA0  
; -- end low active output  
; STA RegPA0  
; macend  
  
clear_v2ok macro  
; -- hi active output: for hi active output uncomment this portion  
; LDI NOT 04H  
; AND RegPA0  
; -- end hi active output  
; -- low active output: for low active output uncomment this portion  
; LDI 04H  
; OR RegPA0  
; -- end low active output  
; STA RegPA0  
; macend
```



# AppNote 29

```
-----  
; Code Offset  
-----  
  
ORG 0  
  
Reset: JMP ResetState  
  
-----  
; Interrupt Handler:  
-----  
Handler:  
    STA    STACK0                ; save ACCU value (ALSO INDEXES IF USED DURING INTERRUPT)  
    STI    RegSysCntl1,0001B      ; turn off interrupts  
    LDR    RegIRQ1                ; save interrupt registers  
    STA    SINT1  
    LDR    RegIRQ2  
    STA    SINT2  
    LDR    STACK0                ; restore ACCU  
    RTI                          ; back to main program  
  
-----  
; ResetState: come here from POR and SCR  
; Define peripherals  
; fall through to BadVddState  
-----  
ResetState:  
    LDI    1000B  
    AND    RegSysCntl2            ; check for SCR timeout  
    JPZ    SCRinit               ; jump if SCR timeout  
  
PORinit:                          ; otherwise hardware init -----  
    STI    RegSysCntl1, 1        ; disable test mode  
    CALL   Trim_osc_base_512kHz  ; trim oscillator for 32kHz  
SCRinit:  
    STI    RegSysCntl2,1100B     ; allow sleep and reset WD  
    STI    RegPA0OE,0110B       ; set PA0 & PA3 to input, PA1 & PA2 to output  
    STI    PA0noPDown,1111B     ; PA0-3 pulldowns off  
    STI    RegPACnt14,1000B     ; PA5 pulldown off  
  
;----- ENABLE output PA5 - OPEN DRAIN - PUSH-PULL, comment out one line below ]  
; STI    RegPA1,0CH              ; PA5 output, OPEN DRAIN ]  
; STI    RegPA1,04H              ; PA5 output, PUSH-PULL ]  
;----- ]  
  
;----- RESET & V2OK outputs PA1 & PA2 - OPEN DRAIN - PUSH-PULL, comment out 3 lines below ]  
; STI    PA0NchOpenDr,0111B     ; PA1 Reset & PA2 v2ok, both OPEN DRAIN ]  
; STI    PA0NchOpenDr,0101B     ; PA1 Reset OPEN DRAIN, PA2 v2ok, PUSH-PULL ]  
; STI    PA0NchOpenDr,0011B     ; PA1 Reset PUSH-PULL, PA2 v2ok, OPEN DRAIN ]  
; STI    PA0NchOpenDr,0001B     ; PA1 Reset & PA2 v2ok, both PUSH-PULL ]  
; ]  
; note: bit 0 set for no open drain on PA0 because of pullup on input ]  
;----- ]  
  
    STI    RegPACnt12,1100B     ; Wakeup on PA0,PA3 & Interrupt from PA0,PA3  
;----- Pushbutton PA0, Watchdog PA3 - rising edge, falling edge, comment out 3 lines below ]  
; STI    RegPaCnt11,1000B       ; PA0 Pbutton & PA3 WDI, both rising edge ]  
; ]  
; STI    RegPaCnt11,1010B       ; PA0 Pbutton rising & PA3 WDI falling edge ]  
; STI    RegPaCnt11,1001B       ; PA0 Pbutton falling & PA3 WDI rising edge ]  
; STI    RegPaCnt11,1000B       ; PA0 Pbutton & PA3 WDI, both falling edge ]  
; note: debounce on for PA0, debounce off for PA3 ]  
;----- ]  
  
    STI    RegSleepCR,1001B     ; turn off PA4 PULL-UP/DOWN, allow sleep  
    STI    RegSVLDlev,BadVddLevel ; Init SVLD level  
    STI    RegCCnt11,0010B     ; prescaler 2kHz, down counter  
    clear_v2ok  
  
; --- end of ResetState --- enter BadVddState
```



```
-----  
; BadVddState  
;   If Vdd is OK go to FirstWDState  
;   otherwise load sleep counter and go to sleep  
-----  
BadVddState:  
    clear_enable                ; set outputs  
    assert_reset  
    STI    EnableCnt,0          ; reset enable count  
    STI    PSaveCnt,0          ; reset power save counter  
    STI    RegSysCntl2,1100B    ; allow sleep and reset WD  
  
; ----- Check V2 and set output  
    STI    RegIRQMask2,0        ; disable SVLD interrupt  
    STI    RegVLDcntl,1000B     ; stop SVLD, enable V2 input  
    STI    RegSVLDlev,V2OKLevel ; Init to V2 level  
    STI    RegVLDcntl,1100B     ; start SVLD  
BADV2busyLoop:  
    NOP  
    LDI    0100B                ; check SVLD busy bit  
    AND    RegVLDcntl  
    JPNZ   BADV2busyLoop        ; jump if still busy  
  
    LDI    1000B                ; check SVLD result bit  
    AND    RegVLDcntl  
    JPNZ   BADV2OKlab          ; V2 is OK  
    clear_v2ok  
    JMP    BADrestartSVLD  
BADV2OKlab:  
    assert_v2ok  
BADrestartSVLD:  
  
; ----- check Vdd with SVLD -----  
    STI    RegIRQMask2,0        ; disable SVLD interrupt  
    STI    RegVLDcntl,0         ; stop SVLD  
    STI    RegSVLDlev,BadVddLevel ; Init SVLD level  
    STI    RegVLDcntl,0100B     ; start SVLD  
SVLDbusyLoop:  
    NOP  
    LDI    0100B                ; check SVLD busy bit  
    AND    RegVLDcntl  
    JPNZ   SVLDbusyLoop        ; jump if still busy  
  
    LDI    1000B                ; check SVLD result bit  
    AND    RegVLDcntl  
    JPNZ   FirstWDState        ; Vdd is OK, start working !  
  
; ----- BADVdd, go to sleep and wait for reset -----  
    STI    RegSysCntl2,1100B     ; allow sleep and reset WD  
    STI    RegSleepCR,1001B     ; turn off PA4 PULL-UP/DOWN, allow sleep 237ms  
    STI    RegSysCntl1,0101B    ; go to SLEEP and wait for reset, returns at Reset:  
  
;--- end of BadVddstate -----
```



```
-----  
; FirstWDState  
; Clear reset  
; Set timer with first timeout  
; wait for first WD input  
; intBadVdd -> BadVddState  
; intWDtimeout -> ClosedWDState  
; intWDI -> ClosedWDState  
; intrRbutton -> ResetPulseState  
-----  
FirstWDState:  
    clear_reset                ; set outputs  
    clear_enable  
    STI    EnableCnt,0          ; reset enable count  
    STI    RegSysCntl2,1100B    ; reset WD  
  
; ----- start counter with TfcCnt  
    STI    RegCCnt12,0          ; stop counter  
    STI    RegCDataL,TfcCnt AND 0FH ; load counter low nibble  
    STI    RegCDataM,(TfcCnt SHR 4) AND 0FH ; load counter middle nibble  
    STI    RegCDataH,(TfcCnt SHR 8) AND 03H ; load counter high nibble  
    STI    RegCCnt12,0001B      ; load counter  
    STI    RegCCnt12,1000B      ; start counter  
  
; ----- Check V2 and set output  
    STI    RegIRQMask2,0        ; disable SVLD interrupt  
    STI    RegVLDCnt1,1000B     ; stop SVLD, enable V2 input  
    STI    RegSVLDlev,V2OKLevel ; Init to V2 level  
    STI    RegVLDCnt1,1100B     ; start SVLD  
FirstV2busyLoop:  
    NOP  
    LDI    0100B                ; check SVLD busy bit  
    AND    RegVLDCnt1  
    JPNZ   FirstV2busyLoop      ; jump if still busy  
  
    LDI    1000B                ; check SVLD result bit  
    AND    RegVLDCnt1  
    JPNZ   FirstV2OKlab         ; V2 is OK  
    clear_v2ok  
    JMP    FirstRestartSVLD  
FirstV2OKlab:  
    assert_v2ok  
FirstRestartSVLD:  
  
; ----- Restart continuous Vdd monitoring  
    STI    RegIRQMask2,0        ; disable SVLD interrupt  
    STI    RegVLDCnt1,0         ; stop SVLD  
    STI    RegSVLDlev,BadVddLevel ; Init SVLD level  
    STI    RegIRQMask1,1011B    ; enable PB, WDI and timeout interrupts  
    STI    RegIRQMask2,0010B    ; enable SVLD interrupt  
    STI    RegVLDCnt1,0110B     ; start SVLD in continuous mode  
    HALT                        ; -FIRST- enable interrupts and enter standby mode  
  
; ----- return here when interrupt  
    LDI    1000B                ; check SVLD result bit: if Vdd passed from OK to Bad  
    AND    RegVLDCnt1          ; during V2 check the interrupt might have been missed  
    JPZ    BadVddState         ; BadVdd -> BadVddState  
    LDR    SINT2                ; load state of RegIRQ2  
    JPNZ   BadVddState         ; intBadVdd -> BadVddState  
    LDR    SINT1                ; load state of RegIRQ1  
    SHRA  
    JPC    ResetPulseState      ; intrRbutton -> ResetPulseState  
; SHRA  
; JPC    ClosedWDState          ; intWDI -> ClosedWDState  
    JMP    ClosedWDState        ; intWDtimeout -> ClosedWDstate  
  
;--- end of FirstWDState -----
```



```
-----  
; ClosedWDstate  
; Clear reset  
; Set timer with closed timeout  
; wait for WD timeout  
; intBadVdd -> BadVddState  
; intWDI -> ResetPulseState  
; intWDtimeout -> OpenWDstate  
; intrbutton -> ResetPulseState  
-----  
ClosedWDstate:  
clear_reset ; set outputs  
STI RegSysCntl2,1100B ; reset WD  
  
; ----- start counter with TcwCnt  
STI RegCCnt12,0 ; stop counter  
STI RegCDataL,TcwCnt AND 0FH ; load counter low nibble  
STI RegCDataM,(TcwCnt SHR 4) AND 0FH ; load counter middle nibble  
STI RegCDataH,(TcwCnt SHR 8) AND 03H ; load counter high nibble  
STI RegCCnt12,0001B ; load counter  
STI RegCCnt12,1000B ; start counter  
  
; ----- assert enable if we have >GoodEnableCnt cycles  
LDI GoodEnableCnt-1 ; compare EnableCnt  
SUB EnableCnt ; carry if EnableCnt < GoodEnableCnt  
JPNC noenable ;  
assert_enable ; enable OK  
JMP checkv2  
noenable:  
clear_enable ; not yet enable, increment in openWDstate  
  
checkv2:  
; ----- Check V2 and set output  
STI RegIRQMask2,0 ; disable SVLD interrupt  
STI RegVLDCnt1,1000B ; stop SVLD, enable V2 input  
STI RegSVLD1lev,V2OKLevel ; Init to V2 level  
STI RegVLDCnt1,1100B ; start SVLD  
V2busyLoop:  
NOP  
LDI 0100B ; check SVLD busy bit  
AND RegVLDCnt1  
JPNZ V2busyLoop ; jump if still busy  
  
LDI 1000B ; check SVLD result bit  
AND RegVLDCnt1  
JPNZ V2OKlab ; V2 is OK  
clear_v2ok  
JMP restartSVLD  
V2OKlab:  
assert_v2ok  
restartSVLD:  
  
; ----- Restart continuous Vdd monitoring  
STI RegIRQMask2,0 ; disable SVLD interrupt  
STI RegVLDCnt1,0 ; stop SVLD  
STI RegSVLD1lev,BadVddLevel ; Init SVLD level  
STI RegVLDCnt1,0110B ; start SVLD in continuous mode  
STI RegIRQMask1,1011B ; enable PB, WDI and timeout interrupts  
STI RegIRQMask2,0010B ; enable SVLD interrupt  
HALT ; -CLOSED- enable interrupts and enter standby mode  
  
; ----- return here when interrupt  
LDI 1000B ; check SVLD result bit: if Vdd passed from OK to Bad  
AND RegVLDCnt1 ; during V2 check the interrupt might have been missed  
JPZ BadVddState ; BadVdd -> BadVddState  
LDR SINT2 ; load state of RegIRQ2  
JPNZ BadVddState ; intBadVdd -> BadVddState  
LDR SINT1 ; load state of RegIRQ1  
SHRA  
JPC ResetPulseState ; intrbutton -> ResetPulseState  
SHRA  
JPC ResetPulseState ; intWDI -> ResetPulseState  
JMP OpenWDstate ; intWDtimeout -> OpenWDstate  
  
;--- end of ClosedWDstate -----
```



```
-----  
; OpenWDstate  
; Clear reset  
; Set timer with open timeout  
; wait for WD input  
; intBadVdd -> BadVddState  
; intWDI -> ClosedWDState  
; intWDtimeout -> ResetPulseState  
; intrRbutton -> ResetPulseState  
-----  
OpenWDState:  
    clear_reset                ; set outputs  
    STI    RegSysCnt12,1100B    ; reset WD  
  
; ----- assert enable if we have >GoodEnableCnt cycles  
    LDI    GoodEnableCnt-1      ; compare EnableCnt  
    SUB    EnableCnt            ; carry if EnableCnt > GoodEnableCnt  
    JPNC   OWnoenable          ;  
    assert_enable              ; enable OK  
    JMP    startctrOpen  
OWnoenable:  
    clear_enable                ; not yet enable  
    INC    EnableCnt            ; increment enable counter  
    STA    EnableCnt  
  
startctrOpen:  
; ----- start counter with TowCnt  
    STI    RegCCnt12,0          ; stop counter  
    STI    RegCDataL,TowCnt AND 0FH ; load counter low nibble  
    STI    RegCDataM,(TowCnt SHR 4) AND 0FH ; load counter middle nibble  
    STI    RegCDataH,(TowCnt SHR 8) AND 03H ; load counter high nibble  
    STI    RegCCnt12,0001B      ; load counter  
    STI    RegCCnt12,1000B      ; start counter  
  
    STI    RegIRQMask1,1011B    ; enable PB, WDI and timeout interrupts  
    STI    RegIRQMask2,0010B    ; enable SVLD interrupt  
    HALT                          ; -OPEN- enable interrupts and enter standby mode  
; ----- return here when interrupt  
    LDR    SINT2                ; load state of RegIRQ2  
    JPNZ   BadVddState          ; intBadVdd -> BadVddState  
    LDR    SINT1                ; load state of RegIRQ1  
    SHRA  
    JPC    ResetPulseState      ; intrRbutton -> ResetPulseState  
    SHRA  
    JPNC   ResetPulseState      ; intWDtimeout -> ResetPulseState  
    STI    PSaveCnt,0           ; reset power save counter  
    JMP    ClosedWDState        ; intWDI -> ClosedWDState  
  
;--- end of OpenWDstate -----
```



```
-----  
; ResetPulseState  
;   Clear Enable  
;   Set reset  
;   Set timer with closed timeout  
;   wait for timeout  
;       intBadVdd      -> BadVddState  
;       intRbutton     -> ResetPulseState  
;       intWDtimeout & SleepCnt<=5  -> FirstWDstate  
;       intWDtimeout & SleepCnt>5   -> PowerSaveState  
-----  
ResetPulseState:  
    clear_enable  
    assert_reset                ; set outputs  
    STI   EnableCnt,0           ; reset enable counter  
    STI   RegSysCnt12,1100B     ; reset WD  
  
nopsave:  
; ----- reset Pushbutton pressed?  
    LDI   0001B                ; check PA0, bit PA[0]  
    AND   RegPA0  
    JPNZ  RPstartcounter       ; reset not pressed, continue  
    STI   RegSysCnt12,1000B     ; reset WD  
    STI   PSaveCnt,0           ; reset power save counter  
    JMP   nopsave              ; reset button pressed, loop here  
  
RPstartcounter:  
; ----- start counter with TrpCnt  
    STI   RegCCnt12,0           ; stop counter  
    STI   RegCDataL,TrpCnt AND 0FH ; load counter low nibble  
    STI   RegCDataM,(TrpCnt SHR 4) AND 0FH ; load counter middle nibble  
    STI   RegCDataH,(TrpCnt SHR 8) AND 03H ; load counter high nibble  
    STI   RegPresc,0100B       ; clear prescaler  
    STI   RegCCnt12,0001B      ; load counter  
    STI   RegCCnt12,1000B      ; start counter  
  
    STI   RegIRQMask1,1001B    ; enable PB and timeout interrupts  
    STI   RegIRQMask2,0010B    ; enable SVLD interrupt  
    HALT                        ; -RPULSE- enable interrupts and enter standby mode  
; ----- return here when interrupt  
    LDR   SINT2                ; load state of RegIRQ2  
    JPNZ  BadVddState          ; intBadVdd -> BadVddState  
    LDR   SINT1                ; load state of RegIRQ1  
    SHRA  
    JPC   ResetPulseState      ; intRbutton -> ResetPulseState  
                                ; intWDtimeout -> ClosedWDstate or PowerSaveState according to PSaveCnt  
    LDI   EnterPowerSaveCnt-1  ; compare PSaveCnt  
    SUB   PSaveCnt             ; carry if PSaveCnt > EnterPowerSaveCnt  
    JPC   PowerSaveState       ; off to power saving  
    INC   PSaveCnt             ; increment power save counter  
    STA   PSaveCnt  
    JMP   FirstWDstate         ; intWDtimeout -> FirstWDstate  
  
;--- end of ResetPulseState -----
```



```
-----  
; PowerSaveState  
; Clear reset  
; Clear enable  
; Set timer with power save timeout  
; wait for timeout  
; intBadVdd -> BadVddState  
; intWDI -> ClosedWDState  
; intWDtimeout -> ResetPulseState  
; intrbutton -> ResetPulseState  
-----  
PowerSaveState:  
    clear_enable  
    clear_reset ; set outputs  
    STI EnableCnt,0 ; reset enable counter  
    STI RegSysCntl2,1100B ; reset WD  
  
; ----- start counter with TpsCnt  
    STI RegCCntl2,0 ; stop counter  
    STI RegCDataL,TpsCnt AND 0FH ; load counter low nibble  
    STI RegCDataM,(TpsCnt SHR 4) AND 0FH ; load counter middle nibble  
    STI RegCDataH,(TpsCnt SHR 8) AND 03H ; load counter high nibble  
    STI RegCCntl2,0001B ; load counter  
    STI RegCCntl2,1000B ; start counter  
  
; ----- Check V2 and set output  
    STI RegIRQMask2,0 ; disable SVLD interrupt  
    STI RegVLDCnt1,1000B ; stop SVLD, enable V2 input  
    STI RegSVLDlev,V2OKLevel ; Init to V2 level  
    STI RegVLDCnt1,1100B ; start SVLD  
PSV2busyLoop:  
    NOP  
    LDI 0100B ; check SVLD busy bit  
    AND RegVLDCnt1  
    JPNZ PSV2busyLoop ; jump if still busy  
  
    LDI 1000B ; check SVLD result bit  
    AND RegVLDCnt1  
    JPNZ PSV2OKlab ; V2 is OK  
    clear_v2ok  
    JMP PSrestartSVLD  
PSV2OKlab:  
    assert_v2ok  
  
PSrestartSVLD:  
; ----- Restart continuous Vdd monitoring  
    STI RegIRQMask2,0 ; disable SVLD interrupt  
    STI RegVLDCnt1,0 ; stop SVLD  
    STI RegSVLDlev,BadVddLevel ; Init SVLD level  
    STI RegVLDCnt1,0110B ; start SVLD in continuous mode  
    STI RegIRQMask1,1011B ; enable PB, WDI and timeout interrupts  
    STI RegIRQMask2,0010B ; enable SVLD interrupt  
    HALT ; -PSAVE- enable interrupts and enter standby mode  
  
; ----- return here when interrupt  
    LDI 1000B ; check SVLD result bit: if Vdd passed from OK to Bad  
    AND RegVLDCnt1 ; during V2 check the interrupt might have been missed  
    JPZ BadVddState ; BadVdd -> BadVddState  
    LDR SINT2 ; load state of RegIRQ2  
    JPNZ BadVddState ; intBadVdd -> BadVddState  
    LDR SINT1 ; load state of RegIRQ1  
    SHRA  
    JPC ResetPulseState ; intrbutton -> ResetPulseState  
    SHRA  
    JPC ClosedWDstate ; intWDI -> ClosedWDstate  
    JMP ResetPulseState ; intWDtimeout -> ResetPulseState  
  
;--- end of PowerSaveState -----
```



## AppNote 29

```
-----  
; Code Offset for trim code  
; ONLY NEEDED FOR SIMULATOR  
-----  
; ORG Trim_osc_base_512kHz  
; STI RegOscTrimH,8H  
; STI RegOscTrimL,8H  
; RET  
-----  
; END  
-----
```

EM Microelectronic-Marin SA (EM) makes no warranty for the use of its products, other than those expressly contained in the Company's standard warranty which is detailed in EM's General Terms of Sale located on the Company's web site. EM assumes no responsibility for any errors which may appear in this document, reserves the right to change devices or specifications detailed herein at any time without notice, and does not make any commitment to update the information contained herein. No licenses to patents or other intellectual property of EM are granted in connection with the sale of EM products, expressly or by implications. EM's products are not authorized for use as components in life support devices or systems.

© EM Microelectronic-Marin SA, 08/05, Rev. A