AN5187

Application note

The BlueNRG-1, BlueNRG-2 improving robustness

Introduction

This document describes some techniques to make applications based on the BlueNRG-1, BlueNRG-2 devices more robust against common problems observed on the field. Here are some suggestions regarding PCB schematics and some about the application firmware configuration, best practices and debugging guidelines.

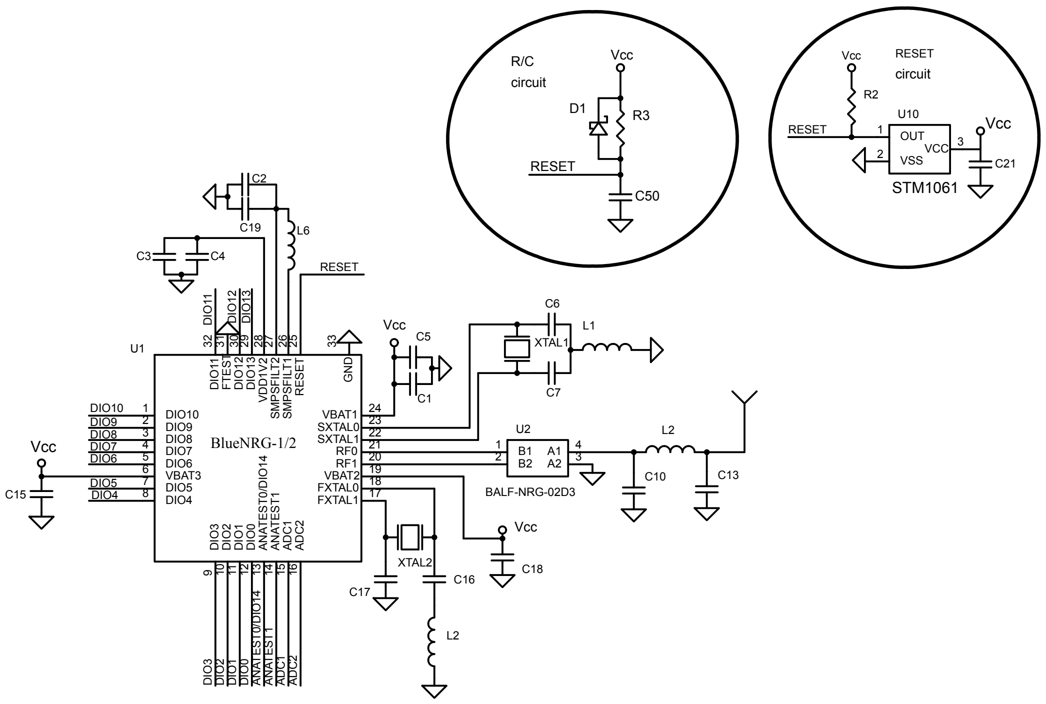
1 Hardware robustness guidelines

1.1 RESET pin control circuits

Two different circuits can be used to control RESET pin of the BlueNRG-1, BlueNRG-2 devices:

1. R/C circuit
2. RESET IC circuit

Figure 1. R/C circuit and RESET IC circuit



1.1.1 R/C circuit

As already seen in the BlueNRG-1, BlueNRG-2 datasheets, the BlueNRG-1, BlueNRG-2 devices need to have a stable power supply before reset is released. In order to achieve this target, an R/C circuit is proposed.

In addition to this, a low drop voltage diode (e.g. Schottky) is added to allow the capacitor to be discharged fast in case of a sudden drop on VDD.

The R/C circuit time constant is a function of resistance and capacitor (τ = RC).

R and C have to be chosen according to the speed the supply voltage rises with:

- Usually R = 100 kohm and C = 10 nF are used

With this values the time constant is 1 ms.

This means:

- After 1 ms (τ) => 63% of the final voltage is reached
- After 2 ms (2 τ) => 86.3% of the final voltage is reached
- After 3 ms (3 τ) => 94.93% of the final voltage is reached

Figure 2. % of final voltage vs time constants

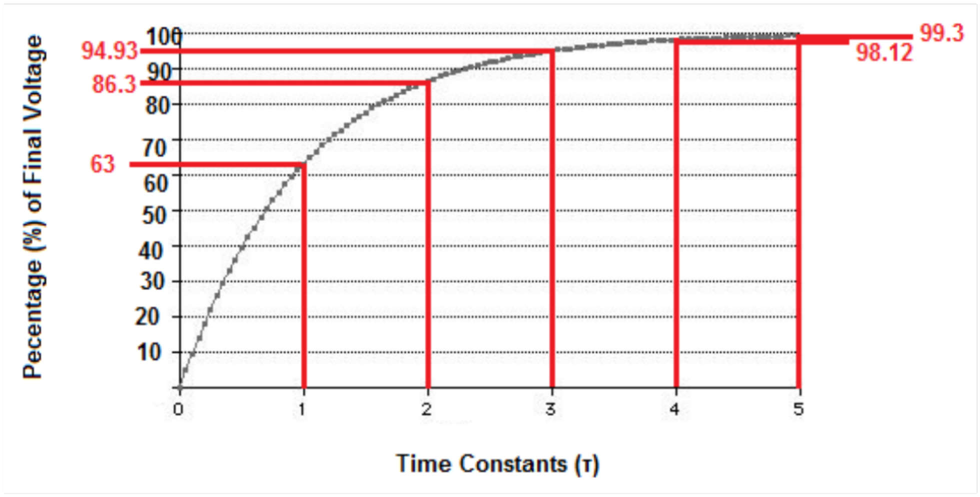
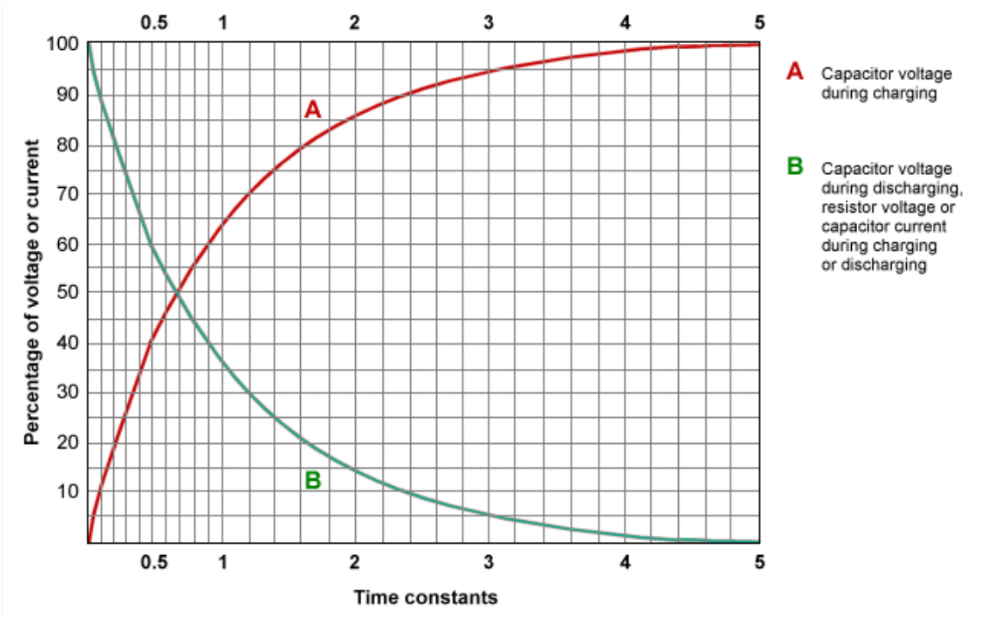


Figure 3. % of voltage or current vs time constants



1.1.2 RESET IC circuit

In alternative to the R/C circuit, a RESET IC can be used to improve application robustness. A RESET IC guarantees that RESET pin is not released until the VDD reaches the minimum operating voltage, thus it can work with very slow VDD ramp. It also allows the device to be kept under reset when voltage drops below operating conditions, so unpredictable device operations are avoided.

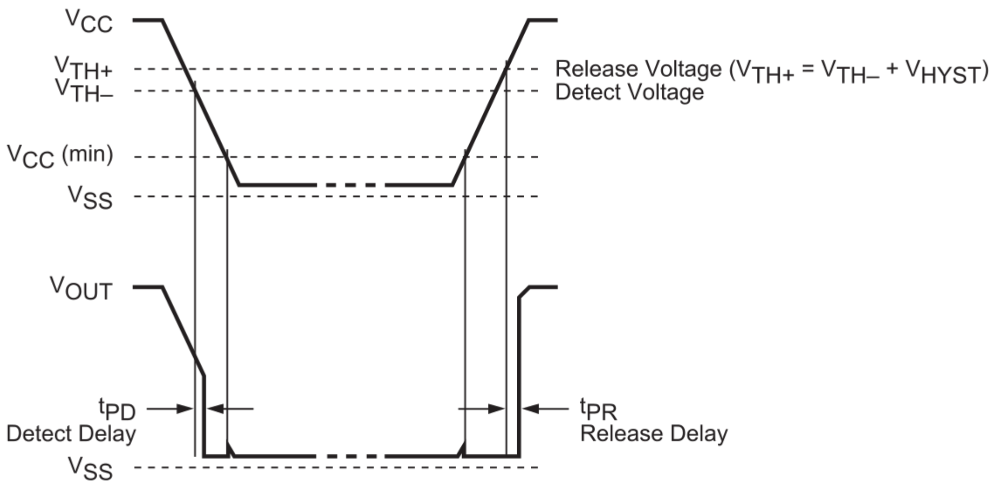
The device proposed is the low power voltage detector STM1061.

The STM1061 low power voltage detector provides monitoring of battery, power supply and regulated system voltage from 1.6 to 5V in 100 mV increments.

- OUT pin is forced low when V_{CC} falls below the voltage detector threshold (V_{TX-})
- OUT pin stays asserted until V_{CC} goes above the voltage detect release (V_{TH+}) + a release delay time (t_{PR}). The release delay time is of 30 us typical
- Voltage detect release (V_{TH+}) is equal to the voltage detector threshold (V_{TX-}) + voltage hysteresis (V_{HYST})

- Output voltage is guaranteed valid down to V_{CC} = 0.7 V
- A pull-up resistor is necessary from OUT pin to any supply voltage
- A 10 k pull-up resistor is sufficient

Figure 4. **STM1061 output voltage vs V_{CC}**



2 Software robustness guidelines

2.1 How to configure firmware using preprocessor #define

In order to properly configure the user application running on the BlueNRG-1, BlueNRG-2 devices, the user must refer to the preprocessor #defines described in the programming manual (PM0257) and select the proper values according to his defined PCB.

Table 1. **Options for HS, LS, SMPS configurations**

Preprocessor option	Preprocessor option values	Description
HS_SPEED_XTAL	*HS_SPEED_XTAL_16MHZ*	High speed crystal configuration: 16 MHz (default configuration)
HS_SPEED_XTAL	*HS_SPEED_XTAL_32MHZ*	High speed crystal configuration: 32 MHz
LS_SOURCE	*LS_SOURCE_EXTERNAL_32kHZ*	Low speed crystal source: external 32 kHz oscillator (default configuration)
LS_SOURCE	*LS_SOURCE_INTERNAL_RO*	Low speed crystal source: internal RO
SMPS_INDUCTOR	*SMPS_INDUCTOR_10uH*	Enable SMPS with 10 uH (default configuration)
SMPS_INDUCTOR	*SMPS_INDUCTOR_4_7uH*	Enable SMPS with 4.7 uH inductor
SMPS_INDUCTOR	*SMPS_INDUCTOR_NONE*	Disable SMPS

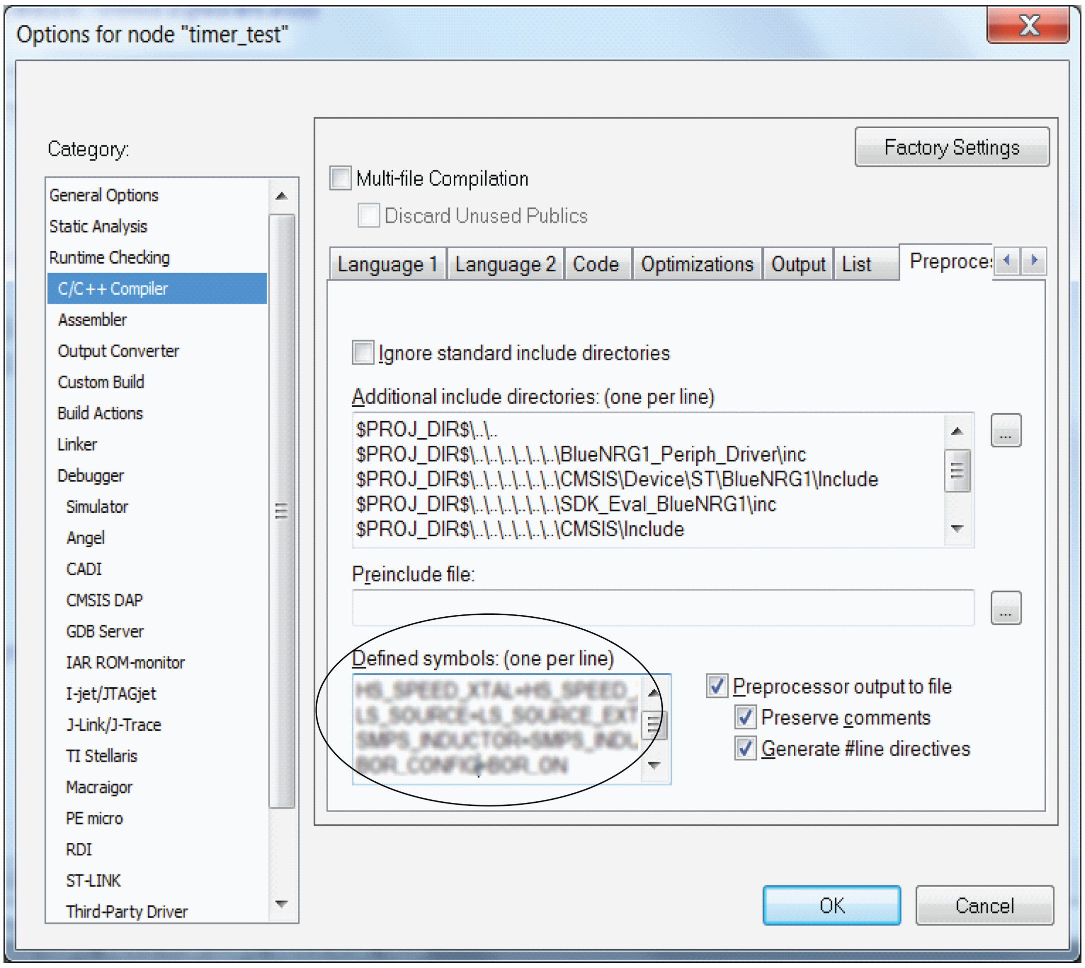
The brown-out-reset (BOR) capability can be enabled/disabled using the following preprocessor option:

Table 2. **BOR configuration**

Preprocessor option	Preprocessor option values	Description
BOR_CONFIG	*BOR_ON*	Enable the BlueNRG-1, BlueNRG-2 BOR (default configuration)
BOR_CONFIG	*BOR_OFF*	Disable the BlueNRG-1, BlueNRG-2 BOR

The preprocessor options can be defined through the supported IDE toolchains

Figure 5. Using preprocessor #define in IAR



Multi-file Compilation [ ]
Discard Unused Publics [ ]
Ignore standard include directories [ ]

Additional include directories: (one per line)

```
$PROJ_DIR$\..\..
$PROJ_DIR$\..\..\..\..\..\..\BlueNRG1_Periph_Driver\inc
$PROJ_DIR$\..\..\..\..\..\..\CMSIS\Device\ST\BlueNRG1\Include
$PROJ_DIR$\..\..\..\..\..\..\SDK_Eval_BlueNRG1\inc
$PROJ_DIR$\..\..\..\..\..\..\CMSIS\Include
```

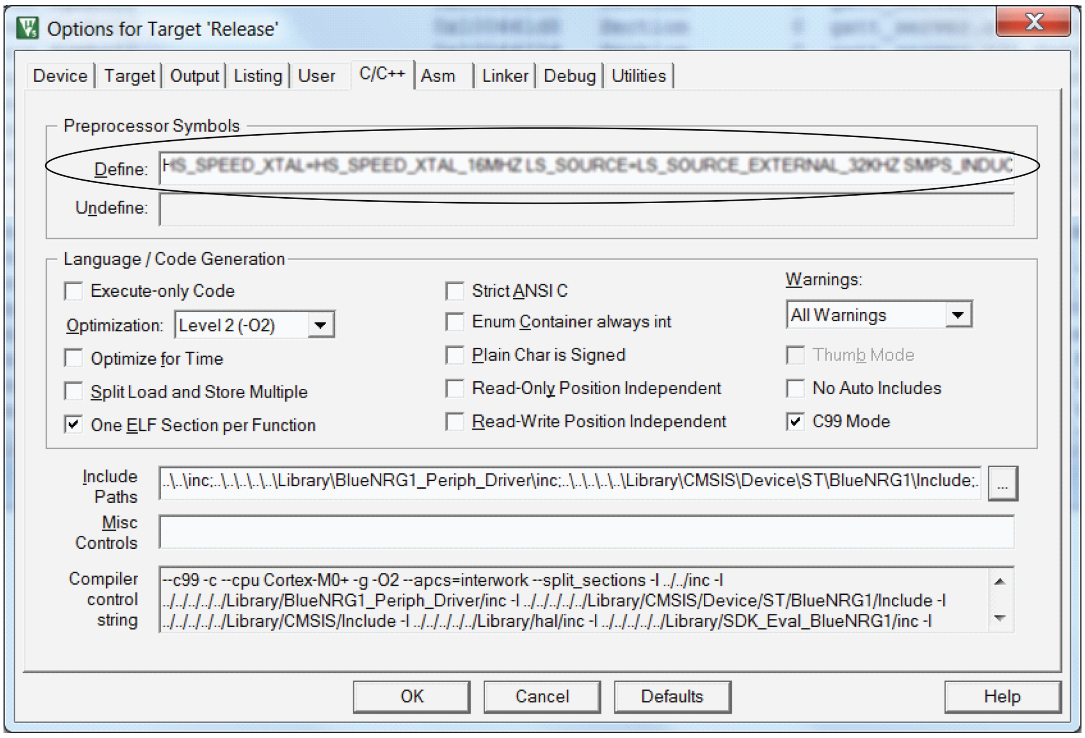
Preinclude file:

Defined symbols: (one per line)

```
HS_SPEED_XTAL=HS_SPEED_
LS_SOURCE=LS_SOURCE_EXT
SMPS_INDUCTOR=SMPS_INDU
BOR_CONFIG=BOR_ON
```

Preprocessor output to file [✓]
Preserve comments [✓]
Generate #line directives [✓]

Figure 6. Using preprocessor #define in KEIL



Define: HS_SPEED_XTAL=HS_SPEED_XTAL_16MHZ LS_SOURCE=LS_SOURCE_EXTERNAL_32KHZ SMPS_INDUC

Undefine:

- [ ] Execute-only Code
- Optimization: Level 2 (-O2)
- [ ] Optimize for Time
- [ ] Split Load and Store Multiple
- [✓] One ELF Section per Function
- [ ] Strict ANSI C
- [ ] Enum Container always int
- [ ] Plain Char is Signed
- [ ] Read-Only Position Independent
- [ ] Read-Write Position Independent
- Warnings: All Warnings
- [ ] Thumb Mode
- [ ] No Auto Includes
- [✓] C99 Mode

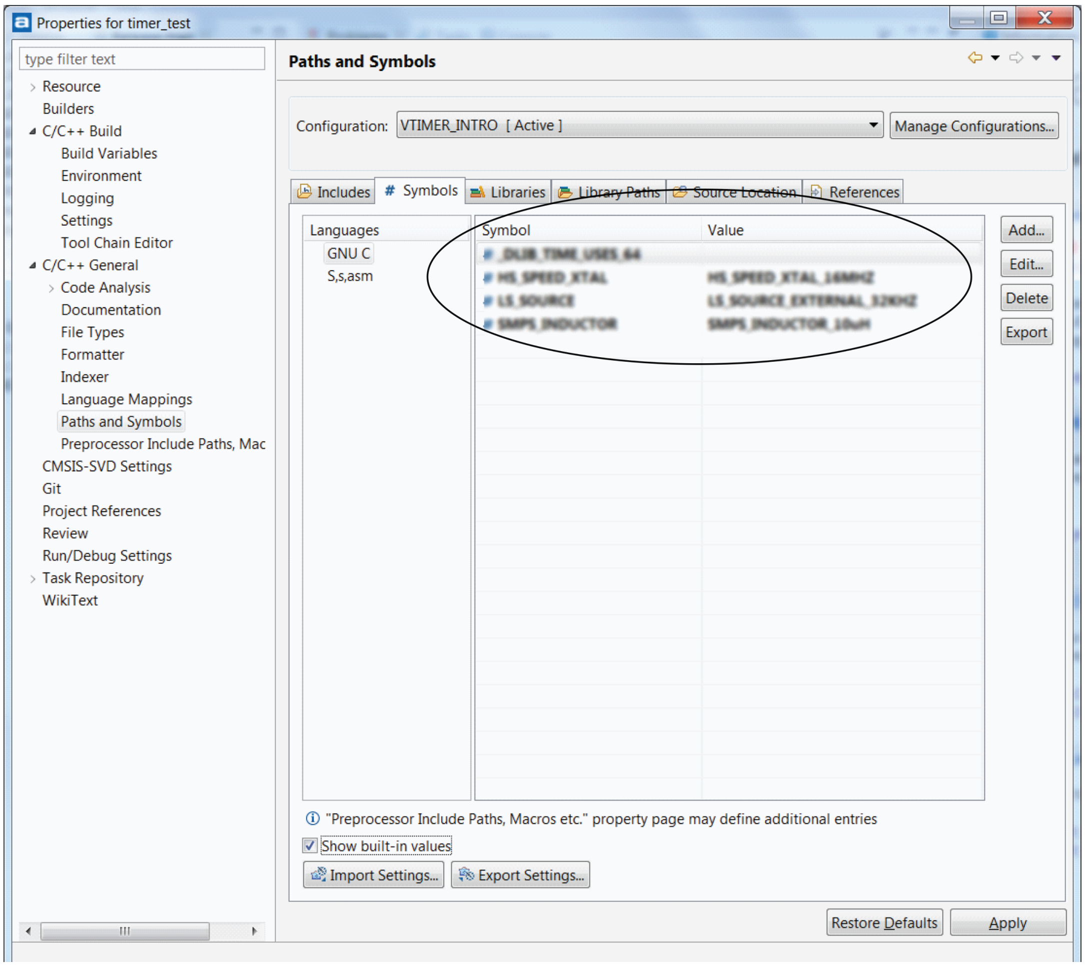
Include Paths: ..\..\inc;..\..\..\..\..\Library\BlueNRG1_Periph_Driver\inc;..\..\..\..\..\Library\CMSIS\Device\ST\BlueNRG1\Include;.

Misc Controls:

Compiler control string:

```
--c99 -c --cpu Cortex-M0+ -g -O2 --apcs=interwork --split_sections -I ../../inc -I
../../../../../Library/BlueNRG1_Periph_Driver/inc -I ../../../../../Library/CMSIS/Device/ST/BlueNRG1/Include -I
../../../../../Library/CMSIS/Include -I ../../../../../Library/hal/inc -I ../../../../../Library/SDK_Eval_BlueNRG1/inc -I
```

Figure 7. **Using preprocessor #define in Atollic**



Properties for timer_test

type filter text

- Resource
- Builders
- C/C++ Build
  - Build Variables
  - Environment
  - Logging
  - Settings
  - Tool Chain Editor
- C/C++ General
  - Code Analysis
  - Documentation
  - File Types
  - Formatter
  - Indexer
  - Language Mappings
  - Paths and Symbols
  - Preprocessor Include Paths, Mac
- CMSIS-SVD Settings
- Git
- Project References
- Review
- Run/Debug Settings
- Task Repository
- WikiText

Paths and Symbols

Configuration: VTIMER_INTRO [Active] | Manage Configurations...

Includes | # Symbols | Libraries | Library Paths | Source Location | References

Languages: GNU C, S,s,asm

Symbol	Value
_DUB_TIME_USES_64	
HS_SPEED_XTAL	HS_SPEED_XTAL_16MHZ
LS_SOURCE	LS_SOURCE_EXTERNAL_32KHZ
SMPS_INDUCTOR	SMPS_INDUCTOR_10uH

Add... | Edit... | Delete | Export

"Preprocessor Include Paths, Macros etc." property page may define additional entries

✓ Show built-in values

Import Settings... | Export Settings...

Restore Defaults | Apply

It is recommended to avoid, unless absolutely necessary, to modify the files:

- system_bluenrg1.[ch]
- system_bluenrg2.h
- sleep.c
- context_switch.s
- Linker files

2.2 BOR management

Brown-out-reset (BOR) is a hardware functionality that allow the device to be forced under reset when voltage drops below minimum operating conditions. As such, the device cannot do unpredictable operations leading to undesired behavior.

To improve robustness of the customer application, BOR should be enabled.

In order to enable BOR, just set *BOR_CONFIG=BOR_ON* in the preprocessor option of your project (refer to Section 2.1 How to configure firmware using preprocessor #define).

2.3 Watchdog management

The proper usage of watchdog allows application to catch software errors where application goes into an infinite loop and it allows them to be recovered by resetting the device.

In order to use this functionality, the best way is to add the watchdog reload to the main loop (and not to the WDG IRQ handler: *WDG_Handler()*).

User should follow these steps:

- Add a watchdog initialization function to be called before entering on main loop (where*WATCHDOG_TIME* is a timeout defined based on the application scenario):

```
void WDG_Init(void)
{
     SysCtrl_PeripheralClockCmd(CLOCK_PERIPH_WDG,
     ENABLE);
     WDG_SetReload(RELOAD_TIME(WATCHDOG_TIME));
     WDG_Enable();
 }
```

- Just call the reload function inside the main loop:

```
while(1) {
     BTLE_StackTick();
     APP_Tick();
     BlueNRG_Sleep(...);
     WDG_SetReload(RELOAD_TIME(WATCHDOG_TIME));
}
```

2.4 Hardware Error Event

Many applications do not work when the *hci_hardware_error_event* is raised.

The hardware error event catch unusual/unexpected failures that can be caused by various instability in the system:

- High speed clock start-up failure
- Slow clock failure
- BTLE stack firmware failures

During the normal operation, hardware error event should not occur, but it is a good habit, in order to improve robustness of the application, to call a system reset in case this event is triggered

```
void hci_hardware_error_event(uint8_t Hardware_Code)
{
  NVIC_SystemReset();
}
```

The *Hardware_Code* is used to indicate some implementation specific types of hardware failures for the controller.

2.5 Crash handling

Cortex M0 has a special handler called hard fault hander that can help catch some firmware issues. Some use cases can be:

- Testing a system, which has random software crashes (may be after several hours and with sleep enabled) without debugging connection
- The application on the field can report unexpected crashes via connection to mobile phone

By setting appropriate handlers in the BlueNRG-1, BlueNRG-2 application, users can catch not only software crashes and reboot the system, but can also report them.

2.5.1 Cortex M0 hard fault

The following figure describes the common Cortex M0 supported hard faults:

Table 3. **Cortex M0 hard faults**

Fault cause	Fault exception	Notes
Vector read error	HardFault	Bus error returned when reading the vector table entry
Fault escalation	HardFault	An SVCall occurred, and the handler group priority is lower or equal to the execution group priority
Memory fault on exception entry stack memory operations	HardFault	Bus error resulting from failure when saving context through hardware
Memory fault on exception return stack memory operations	HardFault	Bus error resulting from failure when restoring context through hardware
Memory fault on instruction access, precise	HardFault	Bus error on an instruction fetch or attempt to execute from memory marked as XN
Precise error on data access	HardFault	Precise bus error because of an explicit memory access
Imprecise error on data access	HardFault	Imprecise bus error because of an explicit memory access
Undefined instruction	HardFault	Unknown instruction
Attempt to execute an instruction when EPSR.T==0	HardFault	Attempt to execute in an invalid EPSR state, for example after a BX type instruction has changed state. This includes state change after entry to or return from exception, or return from interworking instructions
Unaligned load or store	HardFault	This occurs when any load-store instruction attempts to access a non-aligned location

2.5.2 BlueNRG-1,2 crash handler utilities

The BlueNRG-1, 2 crash handler utilities allows the following common hard faults to be detected:

- Catching software errors and sometime hardware issues
- Illegal instructions (e.g. jump to an invalid memory area)
- Access to unaligned memory location
- Access to invalid memory location

Some crash handler utilities/structures are provided on the BlueNRG-1_2 ST SDK package framework (STSW-BLUENRG1-DK):

- Library\hal\inc\crash_handler.h, miscutil.h
- Library\hal\src\miscutil.c

crash_info_t structure defines the overall crash information structure (stack pointer, program counter, registers, ...).

Two utilitiy APIs are provided in order to handle the crash errors.

HAL_GetCrashInfo(&crash_info) API:

- It allows the crash information stored in RAM to be returned, by hard handler, nmi handler and assert handler

HAL_CrashHandler(msp,signature) API:

- It allows the crash information to be stored in RAM and the device to be reset, (crash information can be retrieved by using API *HAL_GetCrashInfo()* API)

2.5.3 Basic example of crash handler utility usage

The following figure describes the main() structure of a BlueNRG-1, BlueNRG-2 application using the crash handler HAL utility *HAL_GetCrashInfo()*.

The following code shows an example of crash handler utility:

```
#include "miscutil.h"

int main(void)
{
  uint32_t counter = 0;
  PartInfoType partInfo;
  crash_info_t crashInfo;

  /* System initialization function */
  SystemInit();

  HAL_GetPartInfo(&partInfo);

  /* Identify BlueNRG1 platform */
  SdkEvalIdentification();

  /* UART initialization */
  SdkEvalComUartInit(UART_BAUDRATE);

  HAL_GetCrashInfo(&crashInfo);
  if ((crashInfo.signature & 0xFFFF0000) == CRASH_SIGNATURE_BASE) {
    printf("Application crash detected\r\n");
    printf("Crash Info signature = 0x%08lx\r\n", crashInfo.signature);
    printf("Crash Info SP        = 0x%08lx\r\n", crashInfo.SP);
    printf("Crash Info R0        = 0x%08lx\r\n", crashInfo.R0);
    printf("Crash Info R1        = 0x%08lx\r\n", crashInfo.R1);
    printf("Crash Info R2        = 0x%08lx\r\n", crashInfo.R2);
    printf("Crash Info R3        = 0x%08lx\r\n", crashInfo.R3);
    printf("Crash Info R12       = 0x%08lx\r\n", crashInfo.R12);
    printf("Crash Info LR        = 0x%08lx\r\n", crashInfo.LR);
    printf("Crash Info PC        = 0x%08lx\r\n", crashInfo.PC);
    printf("Crash Info xPSR      = 0x%08lx\r\n", crashInfo.xPSR);
  }
  /* infinite loop */
  while(1)
  {
    if (counter == 0 ) {
      printf("Hello World: BlueNRG-%d (%d.%d) is here!\r\n",
             partInfo.die_id,
             partInfo.die_major,
             partInfo.die_cut);
    }
    counter = (counter +1) % 0xFFFFF;

  }
}
```

Where:

- *crash_info* variable defines the variable where crash info information is stored
- *HAL_GetCrashInfo(&crash_info)* allows the crash information to be got
- At reset, if an assert occurs, application detects it (through the *if (crash_info.signature&0xFFFF0000) == CRASH_SIGNATURE_BASE)*)

By analyzing the *crash_info* data (registers, stack pointer, programm counter value), the user can get some useful information in order to identify the root cause.

The following code shows how to handle the NMI and hard fault exception crash information:

```
/**
  * @brief  This function handles NMI exception.
  */
NOSTACK_FUNCTION(NORETURN_FUNCTION(void NMI_Handler(void)))
{
  HAL_CrashHandler(__get_MSP(), NMI_SIGNATURE);
  /* Go to infinite loop when NMI exception occurs */
  while (1)
  {}
}

/**
  * @brief  This function handles Hard Fault exception.
  */
NOSTACK_FUNCTION(NORETURN_FUNCTION(void HardFault_Handler(void)))
{
  HAL_CrashHandler(__get_MSP(), HARD_FAULT_SIGNATURE);
  /* Go to infinite loop when Hard Fault exception occurs */
  while (1)
  {}
}
```

This function allows respectively, NMI and hard fault exception crash information to be stored, on related interrupt handlers:

- *HAL_CrashHandler(__get_MSP(), NMI_SIGNATURE);* on BlueNRG-1,2 *NMI_Handler()* irq handler
- *HAL_CrashHandler(__get_MSP(), HARD_FAULT_SIGNATURE);* on BlueNRG-1,2 *HardFault_Handler()* irq handler

2.6 Debug interface disabled: how to recover?

As declared on the BlueNRG-1, BlueNRG2 datasheets, there are certain situations where debug access is disabled and the chip cannot be accessed, including:

- The application that disables debug pins
- The application that sets the device in sleep or standby state, in which the debug port is not powered

These cases are common during the application development and the device can end up in a state where the debug access is no longer possible. To recover this situation, it is recommended to force IO7 pin high and hardware reset so to cause the execution of the ROM bootloader code. The user can then connect with SWD interface and erase the device Flash memory.

2.7 Tips for debugging under sleep

The following tips can be applied to allow an application to be debugged when enabling low power mode:

1. Disable deep sleep by using in the API BlueNRG_Sleep() the value SLEEPMODE_CPU_HALT instead of SLEEPMODE_NOTIMER or SLEEPMODE_WAKETIMER as follows:
   – *BlueNRG_Sleep(SLEEPMODE_CPU_HALT, 0, 0)*
     In this mode, CPU never goes to sleep and debug access is always granted. The application has to be modified and execution time may be different.
2. Use a GPIO to wake up the system and grant access to the debugger. The *BlueNRG_Sleep()* API should be changed to enable wakeup from the selected wakeup source as follows:
   – */* Example for wakeup on IO13 low level */*
     BlueNRG_Sleep(SLEEPMODE_NOTIMER, WAKEUP_IO13,(WAKEUP_IOx_LOW << WAKEUP_IO13_SHIFT_MASK));

When the debugger access is wanted, the user should force the wakeup I/O (IO13 in the example) low and then use the “Attach to running target” in the debugger.

2.8 Key RAM variables

The BlueNRG-1, BlueNRG-2 applications associated to the BlueNRG-1_2 ST SDK (STSW-BLUENRG1-DK) define a set of key variables in RAM, which are preserved through the associated linker files and SDK framework.

The variables are:

- ota_sw_activation (address: 0x20000004)
  - OTA service manager activation flag (jump to OTA service manager)
- savedMSP (address: 0x20000008)
  - Private storage to hold the saved stack pointer
- wakeupFromSleepFlag (address: 0x2000000C)
  - A wakeup from standby or sleep occurred
- app_base (address: 0x20000010)
  - The application base address. Used by OTA IRQ stub file to determine the effective application base address and jump to the proper IRQ handler
- flash_sw_lock (address: 0x20000014)
  - This is used to lock/unlock the flash operations in software
- rfTimeout (address: 0x20000018)
  - Timeout for next RF transaction
- BOR_config (address: 0x2000001C)
  - BOR configuration storage
- __blueflag_RAM (address: 0x20000030)
  - Flag for network coprocessor updater mode activation
- crash_info_ram (address: 0x20000034)
  - Crash handler storage information block to store register information where an assert is verified

It is strictly recommended to avoid removing/modifying these variables and related addresses.

3 List of acronyms

Table 4. List of acronyms used in the document

Term	Meaning
BLE	Bluetooth low energy
FW	Firmware
GAP	Generic access profile
HW	Hardware
SDK	Software development kit
SW	Software

4 References

Table 5. References

Name	Description
www.st.com/bluenrg-1	BlueNRG-1 web page with device datasheet, SW packages, kits and documentation
www.st.com/bluenrg-2	BlueNRG-2 web page with device datasheet, SW packages, kits and documentation
STSW-BLUENRG1-DK	BlueNRG_1, BlueNRG-2 SW package with BLE stack v2.x

Revision history

Table 6. **Document revision history**

Date	Revision	Changes
19-Oct-2018	1	Initial release
07-Feb-2019	2	Updated Section 1.1.1 R/C circuit and Section 2.8 Key RAM variables.
03-Mar-2020	3	Updated Section 2.8 Key RAM variables.

Contents