Application overview

This application is based on the STM8S-DISCOVERY. It demonstrates how to use the STM8S window watchdog (WWDG) and independent watchdog (IWDG) in conjunction with the clock security system (CSS) to handle software and hardware failures.

Two external pushbuttons are used to simulate malfunctions while the LEDs monitor the application progress.

Once the STM8S105C6T6 is powered up through an USB cable connected to the host PC, LED LD2 starts blinking meaning that the programming operation has completed successfully.

Reference documents

- STM8S-DISCOVERY evaluation board user manual (UM0817).
- “Developing and debugging your STM8S-DISCOVERY application code” user manual (UM0834).
- STM8S105xx datasheet
- STM8S reference manual (RM0016)
Contents

1 Application description ............................................. 5
   1.1 Hardware requirements ........................................ 5
   1.2 Application schematics ....................................... 6
   1.3 Application principle ....................................... 6

2 Software description .................................................. 9
   2.1 STM8S peripheral configuration ............................... 9
      2.1.1 WWDG .................................................. 9
      2.1.2 Clock security system (CSS) ............................ 9
      2.1.3 IWDG .................................................. 9
      2.1.4 CLK .................................................. 10
      2.1.5 GPIOs ............................................... 10
      2.1.6 EXTI ................................................ 10
      2.1.7 Data EEPROM .......................................... 10
      2.1.8 RST .................................................. 10
      2.1.9 CCO ................................................ 10
   2.2 Exclusion of the Standard STM8S standard firmware library ...... 10
   2.3 Application software flowchart .................................. 11
      2.3.1 Main loop flowchart .................................... 11
      2.3.2 Interrupt function flowcharts ......................... 12
      2.3.3 State machine flowchart ................................ 12
      2.3.4 Reset handling function flowchart ..................... 15
      2.3.5 Saving SystemState in data EEPROM ................... 16

3 Revision history .......................................................... 18
List of tables

Table 1. List of external components .................................................. 5
Table 2. Interrupt routine versus flag .................................................. 12
Table 3. Document revision history ..................................................... 18
List of figures

Figure 1. Application schematics ................................................................. 6
Figure 2. Application state machine ............................................................ 8
Figure 3. WWDG software monitoring ........................................................... 9
Figure 4. Main loop flowchart ................................................................. 11
Figure 5. State_Machine() function flowchart ........................................ 14
Figure 6. Reset handling routine flowchart ................................................ 15
Figure 7. Creating a segment in Zero Page with STVD ................................ 17
1 Application description

1.1 Hardware requirements

No on-board resources are required. 

*Table 1* gives the list of the external components used by the application.

<table>
<thead>
<tr>
<th>Component name</th>
<th>Value</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD2, LD3, LD4, LD5</td>
<td>-</td>
<td>Standards LEDs</td>
</tr>
<tr>
<td>R2, R3, R4, R5</td>
<td>510 Ω</td>
<td>Protective resistors</td>
</tr>
<tr>
<td>C</td>
<td>100 nF</td>
<td>Debounce filter</td>
</tr>
<tr>
<td>Button1</td>
<td>-</td>
<td>Standard pushbutton</td>
</tr>
<tr>
<td>Button2</td>
<td>-</td>
<td>Standard pushbutton</td>
</tr>
<tr>
<td>C1</td>
<td>470 nF</td>
<td>Capacitor</td>
</tr>
</tbody>
</table>
1.2 Application schematics

*Figure 1* shows how to interface the LEDs and the pushbuttons with the STM8S-DISCOVERY.

For STM8S-DISCOVERY implementation details, refer to the board schematics provided in the STM8S-DISCOVERY user manual (UM0817).

Button1 requires an RC debounce filter to avoid triggering several interrupts. It consists of a capacitor C and PB0 internal pull-up resistor (about 45 KΩ). No pull-up resistor is required for Button1 as the internal pull-up of PBO I/O pin is used.

**Figure 1. Application schematics**

![Application schematic diagram](image)

1.3 Application principle

Two external pushbuttons are used to simulate malfunctions while the LEDs monitor the application progress.

Button1 can be used in two different ways depending on the application context:

1. To demonstrate the WWDG capability for managing software failures such as application freeze:
   - When pressing Button1, the application lengthens the delay between two WWDG refresh operations until this delay becomes too long. The WWDG then triggers a reset.
2. To trigger a software reset.

Button2 is used to demonstrate CSS and IWDG recovery capabilities in case of HSE clock failure. Pressing Button2 connects an additional high value capacitor to the OSCIN pin.
which makes the oscillator lose his balance. This induces a clock disruption thus simulating an HSE oscillator failure. When this kind of hardware failure occurs, the CSS automatically switches the master clock from HSE to HSI. If the CSS is disabled, the IWDG handles the hardware failure since it is clocked by the LSI clock which stays active even when the master clock fails.

The CPU frequency ($f_{CPU}$) and the master clock switching can be monitored by connecting an oscilloscope to CCO (PE0 pin).

LD2, LD3, LD4, and LD5 monitor the current state of the application. At application startup, only LD2 blinks. Each pushbutton event makes the application execution progress as described in Figure 2.

*Figure 2* describes the application principle and the actions performed at each transition.
### Application State Machine

<table>
<thead>
<tr>
<th>Functional Step</th>
<th>LED Status</th>
<th>Actions</th>
<th>Master Clock</th>
</tr>
</thead>
<tbody>
<tr>
<td>Startup or STEP1 complete</td>
<td>YES</td>
<td>LD2 blinks</td>
<td>$f_{HSE} = 16,\text{MHz}$</td>
</tr>
<tr>
<td>STEP1</td>
<td>YES</td>
<td>LD2 blinks slowly</td>
<td>Application runtime lengthened</td>
</tr>
<tr>
<td>Button1 pressed</td>
<td>YES</td>
<td>LD1, LD2, LD3, LD4 blink once</td>
<td>Application runtime lengthened again causing WWDG to trigger a reset</td>
</tr>
<tr>
<td>STEP3 complete</td>
<td>YES</td>
<td>LD3 blinks</td>
<td>HSE enabled and selected as Master clock, CSS enabled</td>
</tr>
</tbody>
</table>
| Button2 pressed | YES | LD3 and LD4 blink once | HSE disabled:  
- CSS triggers an interrupt  
- Master clock is switched to HS/L |
| STEP5 complete | YES | LD3 blink slowly | CSS disabled |
| Button1 pressed | YES | LD3 and LD4 blink twice | Software reset is triggered |
| STEP7 complete | YES | LD4 blinks once | CSS disabled, HSE enabled and selected as master clock |
| Button2 pressed for more than 1s | YES | LD2 and LD5 blink twice | HSE disabled:  
- WWDG not refreshed in time  
- Reset is triggered |

$f_{HSE}$ = 16\,\text{MHz}

$f_{HSE}$ = 2\,\text{MHz}

$f_{HSE}$ = 16\,\text{MHz}

AI 18452
2 Software description

The application software does not use STM8S standard firmware library. It rather consists in optimized code with direct register accesses to control and use the general purpose peripherals as described in Section 2.1.

2.1 STM8S peripheral configuration

2.1.1 WWDG

The WWDG peripheral is used for software monitoring (see Figure 3). The refresh period and the illegal refresh window duration are defined by the user. The microcontroller is reset when the WWDG counter is not refreshed by the end of the refresh period or if it is refreshed during the illegal refresh window.

In this particular example, the WWDG resets the STM8S because of a user simulated software failure causing the WWDG counter not to be refreshed in time. As it depends on the MCU clock, the WWDG is useless when a hardware failure, such as an MCU clock failure, occurs.

Figure 3. WWDG software monitoring

![WWDG refresh period](image)

2.1.2 Clock security system (CSS)

The Clock Security System (CSS) monitors HSE crystal clock source failures. When the HSE clock is selected as master clock (f\text{MASTER}), if the HSE clock fails because of a broken or disconnected resonator (or for any other reason), the clock controller activates a stall safe recovery mechanism by automatically switching the master clock to the auxiliary clock source (f_{\text{HSI/8}}). Once selected, the auxiliary clock source remains enabled until the microcontroller is reset.

2.1.3 IWDG

The IWDG peripheral is used to monitor software and hardware failures. Its counter is refreshed by the application. The microcontroller is reset if the IWDG counter is not refreshed in time because a hardware failure occurred.

In this particular example, the IWDG monitors the HSE clock failure when the CSS is disabled since it is clocked by the 128 kHz LSI internal RC clock source which remains active in case of master clock failure.
2.1.4 **CLK**

The clock control enables and delivers the correct clock frequency to the CPU and peripherals. It configures the HSI and HSE prescalers division factor from 8 to 1.

2.1.5 **GPIOs**

The application drives the MCU I/Os to interface the microcontroller with external hardware components, i.e. the LEDs and the pushbuttons.

2.1.6 **EXTI**

The external interrupt sensitivity is configured to trigger an interrupt each time a falling edge, and only a falling edge, is detected on PB0.

2.1.7 **Data EEPROM**

The `SystemState` variable represents the current state of the application. It is stored in the STM8S105C6T6 data EEPROM thus allowing to save the application progress in the state machine when the microcontroller is reset (see *Figure 5*).

2.1.8 **RST**

The reset register (RST_SR) is used in conjunction with the IWDG and WWDG to determine which of them has triggered a reset.

2.1.9 **CCO**

The Configurable clock output register (CLK_CCOR) allows to select the clock source that is output on the CCO pin.

2.2 **Exclusion of the Standard STM8S standard firmware library**

As this application uses optimized code, the `stm8s.h` file must be modified not to include the STM8S standard firmware library. This is done by commenting the following define statement:

```c
#define USE_STDPERIPH_DRIVER
```
2.3 Application software flowchart

2.3.1 Main loop flowchart

The code main loop initializes the required features, unlocks data EEPROM for writing and calls the functions required to implement the general application algorithm. Figure 4 shows the flowchart of the application software main loop.

Figure 4. Main loop flowchart
2.3.2  Interrupt function flowcharts

Port B interrupt function handler

This interrupt routine is associated to button1. It is called when the user presses button (see Table 2).

Clock interrupt function handler

This interrupt routine is called when a failure is detected on the HSE clock used as master clock (see Table 2).

Table 2.  Interrupt routine versus flag

<table>
<thead>
<tr>
<th>Interrupt routine</th>
<th>Event</th>
<th>Flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>PortB_IRQHandler()</td>
<td>Button1 pressed</td>
<td>ButtonPressed set</td>
</tr>
<tr>
<td>Clk_IRQHandler()</td>
<td>Failure detected on HSE clock and HSE = master clock</td>
<td>CSS_flag set</td>
</tr>
</tbody>
</table>

2.3.3  State machine flowchart

The State_Machine() function implements the algorithm that controls the progress of the application execution according to external events.

1.  $SystemState = 0$
   
   LD2 blinks. The application is ready to demonstrate how to use the WWDG:
   
   a) The first time Button1 is pressed, an interrupt is triggered and the ButtonPressed flag is set (see Section: Port B interrupt function handler). As a result, the execution time is lengthened. This is signalled by LD2 blinking at a slower rate since the $Dly$ parameter which controls the delay between each WWDG refresh is incremented.
   
   b) The second time Button1 is pressed, the delay introduced by $Dly$ becomes too long to refresh the WWDG counter in time. The STM8S105C6T6 is then reset by the WWDG. Just before the reset, $SystemState$ variable is saved in the data EEPROM. This shows how to use the data EEPROM for data saving (see Section 2.3.5: Saving SystemState in data EEPROM).

2.  $SystemState = 1$
   
   LD3 blinks. The HSE crystal clock is selected as the master clock source to demonstrate the effectiveness of the clock failure mechanism performed by the CSS when the clock fails. In this case, the failure is induced when pressing Button2 (see Figure 2: Application state machine).

   However, the application is not stopped as the CSS switches automatically the master clock source from the HSE to the HSI clock. The completion of the switching operation
triggers an interrupt and launches the CLK_IRQHandler() routine. CSS_flag is set and the application goes to step 3 ($SystemState = 2$).

3. **SystemState = 2**

LD3 blinks slower than in state 2 since the master clock source is $f_{HS}/8$.

Pressing Button1 triggers an interrupt. The ButtonPressed flag is set by the PortB_IRQHandler() routine, and a software reset is generated to disable the CSS (see [Section 2.3.2: Interrupt function flowcharts](#)).

$SystemState$ value is saved in the data EEPROM before the reset occurs.

4. **SystemState = 3**

LD4 blinks. The HSE clock is set to the master clock source and the CSS is disabled to demonstrate how the IWDG can be used to solve microcontroller hardware or software failures.

Pressing Button2 for longer than the IWDG refresh window (1.02 s) stops the application by simulating a master clock source failure. After the refresh period has elapsed, the IWDG resets the STM8S105C6T6, and the application is re-initialized ($SystemState = 0$).

*Figure 5* shows the flowchart of the State_Machine() function.
Figure 5. State_Machine() function flowchart
2.3.4 Reset handling function flowchart

When a reset occurs, the application reads the RST_SR register to determine which watchdog has triggered the reset and act accordingly.

When a WWDG reset occurs, the reset handling routine retrieves the SystemState value previously stored by the application in data EEPROM, preventing the state machine from being re-initialized.

SystemState is initialized to 0 at the first execution.

When SystemState equals 1, the CSS is enabled to monitor HSE disruptions (see Section 2.3.3: State machine flowchart).

As soon as the application exits from the IWDG reset, SystemState must be reset to re-initialize the application state machine.

Figure 6 shows the flowchart of the reset handling function.
### 2.3.5 Saving SystemState in data EEPROM

To demonstrate how to use the data EEPROM for saving data, the `SystemState` global variable is saved and then read back from memory.

This step is not mandatory since there are several solutions to keep this variable unchanged during reset. The solution depends on the compiler.

#### Raisonance compiler

`SystemState` must be declared as global since global variables are created either in the Zero Page or in the RAM Data segments and are consequently not re-initialized at reset.

#### Cosmic compiler

Cosmic compiler re-initializes global variables at startup. The steps required to save the value of `SystemState` in data EEPROM:

1. Declare the global variable with the following example syntax
   ```
   #pragma section [my_sec]
   u8 SystemState;
   #pragma section []
   ```
2. Then two solutions are possible:
   - Either create the `my_sec` section in the Zero Page with the extension ‘-ib’ that prevents Cosmic compiler from re-initializing its content at reset. This can be done in the STVD Project > Settings > Linker tab > Category: Input menu (see Figure 7).
   - Or directly modify the project linker file to perform the same operation. It is mandatory to remove the Auto option in STVD linker tab (see Figure 7). To do this, open the file `CSS_WDG_Robustness.lkf` located in the STVD/Cosmic/Debug folder of the Cosmic project and modify it as follows.
     ```
     # Segment Zero Page:
     +seg .bsct -b 0x0 -m 0x100 -n .bsct
     +seg .ubsct -a .bsct -n .ubsct
     +seg .bit -a .ubsct -n .bit -id
     +seg .share -a .bit -n .share -is
     +seg .my_sec -a .share -n .my_sec -ib
     ```
     For more details, refer to the Cosmic user manual available in the Cosmic installation directory.
Figure 7. Creating a segment in Zero Page with STVD
### 3 Revision history

Table 3. Document revision history

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>17-Dec-2010</td>
<td>1</td>
<td>Initial release.</td>
</tr>
</tbody>
</table>
Please Read Carefully:

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST’s terms and conditions of sale. Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST’S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS EXPRESSLY APPROVED IN WRITING BY AN AUTHORIZED ST REPRESENTATIVE, ST PRODUCTS ARE NOT RECOMMENDED, AUTHORIZED OR WARRANTED FOR USE IN MILITARY, AIR CRAFT, SPACE, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS, NOR IN PRODUCTS OR SYSTEMS WHERE FAILURE OR MALFUNCTION MAY RESULT IN PERSONAL INJURY, DEATH, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE. ST PRODUCTS WHICH ARE NOT SPECIFIED AS “AUTOMOTIVE GRADE” MAY ONLY BE USED IN AUTOMOTIVE APPLICATIONS AT USER’S OWN RISK.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.

Information in this document supersedes and replaces all information previously supplied.

The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

© 2010 STMicroelectronics - All rights reserved

STMicroelectronics group of companies
Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Philippines - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

www.st.com