SMART ARM-based Microcontrollers

AT09381: SAM D - Debugging Watchdog Timer Reset

APPLICATION NOTE

Introduction

This application note shows how the early warning interrupt can be used to debug a WDT reset situation.

Table of Contents

1. Prerequisites

This application note requires:

- Atmel® Studio 6.1 or newer
- A SAM D Xplained PRO
- Example code

2. SAM D Watchdog Timer Reset Debugging

The Watchdog Timer (WDT) is used to reset the device to recover from error situations such as runaway or deadlocked code. This appnote does not show how to use or setup the WDT. This appnote shows how it is possible to determine what code was being executed before the WDT resets the device.

This appnote is bundled with an Atmel Studio project that can be used as a reference to the appnote.

This appnote should be applicable to any device with a WDT with the Early Warning Interrupt present.

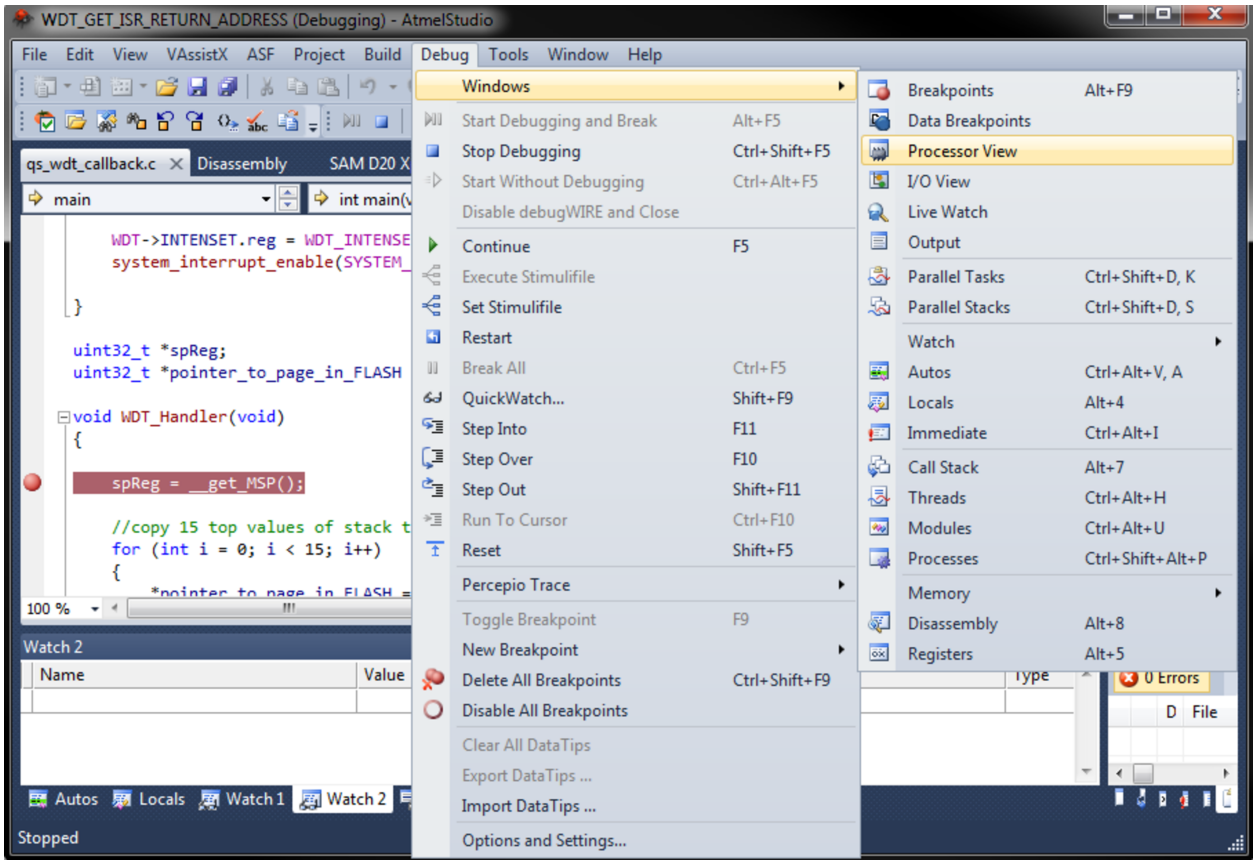
2.1. The Early Warning Interrupt

The WDT in the SAM D has got an Early Warning interrupt. This interrupt can be enabled to indicate an upcoming WDT watchdog time-out condition. This can be used to find out which address the WDT interrupt handler will jump back to when the interrupt handler is done executing.

The address that the interrupt handler jumps back to is the address of the code that is being executed. It may be possible to see if the device is stuck in a loop or something else that keeps it from issuing a reset of the WDT timer.

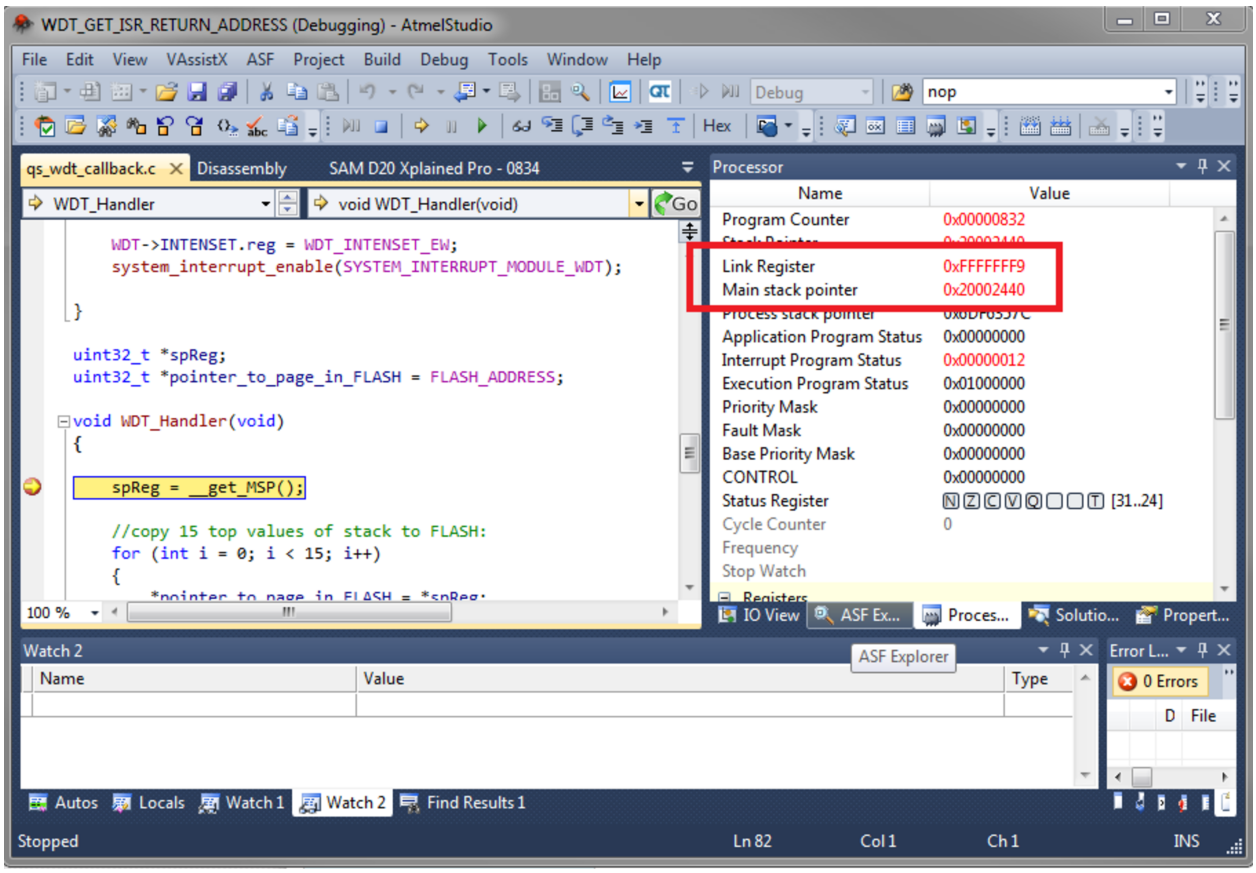
The link register value can be checked by placing a breakpoint inside the interrupt handler. When the program enters the handler, the link register can be read by Atmel Studio by checking the Processor View, see the figure below.

Figure 2-1. Enabling Processor View



When checking the processor view (see the figure below) in Atmel Studio when inside the WDT_Handler() function in the project code, it can be seen that the value of the link register is 0xFFFFFFF9. According to ARM® "Cortex®-M0+ Devices Generic User Guide" under "Home > The Cortex-M0+ Processor > Exception model > Exception entry and return" this means that the return address of the handler will be found in the stack.

Figure 2-2. Processor View



In the processor view we can find the main stack pointer. This points to the top of the stack. It is possible to read out what is in the stack when in a debug session by going to Debug->Windows->Memory->"Memory 1" and checking what is at the address pointed to by the main stack pointer. But in order to debug a device that is not connected to a debugger it is necessary to copy the stack to flash.

This can be done by copying the main stack address to a pointer and then copying the content of the top of the stack. The following WDT_Handler will do this:

```
#define FLASH_ADDRESS 0x3FF80;
uint32_t *spReg;
uint32_t *pointer_to_page_in_flash = FLASH_ADDRESS;

void WDT_Handler(void) {
    spReg = __get_MSP();
    //copy 15 top values of stack to flash:
    for (int i = 0; i < 15; i++) {
        *pointer_to_page_in_flash = *spReg;
        pointer_to_page_in_flash++;
        spReg++;
    }
    //write page buffer to flash:
    NVMCTRL->CTRLA.reg = NVMCTRL_CTRLA_CMDEX_KEY | NVMCTRL_CTRLA_CMD_WP;
```

```
    //Clear EW interrupt flag
    WDT->INTFLAG.bit.EW = 1;
}
```

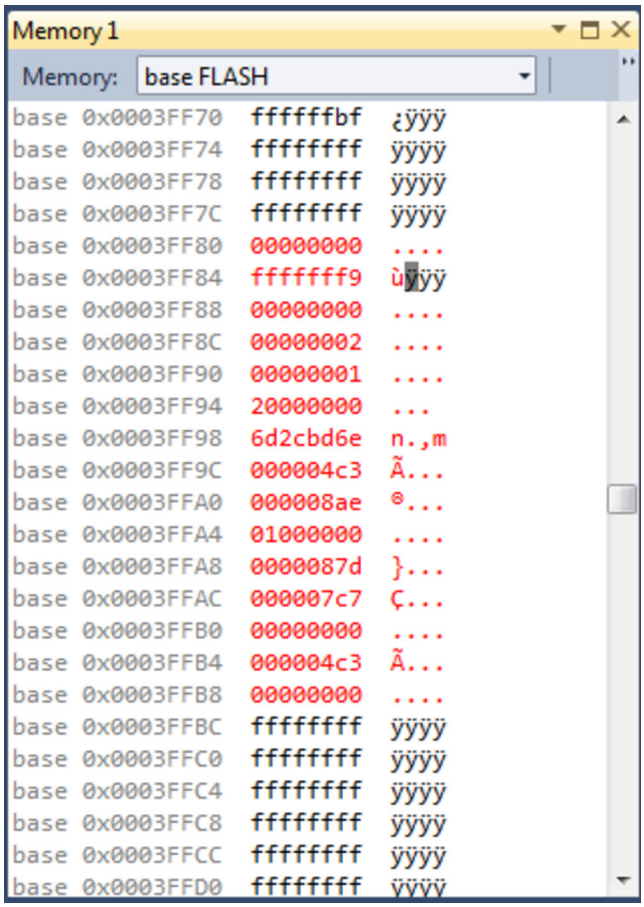
After this code has been executed, the FLASH will be updated at the page starting with address 0x3FF80. In the example project, this flash page is not used by the application code. For a different application or a smaller device the page that is written to should be chosen differently.

It is necessary to check what is written in flash by going to the Memory view (in Atmel Studio) and see what is in flash after the interrupt handler has written to flash. This should be done before the WDT has time to reset the device. In order to do this place a break point in the interrupt handler at the end of the write to flash.

When looking at the Memory view (see the figure below) we can see that some of the content does not look like addresses to flash and some of it looks like it would point to the interrupt vector table. The reason for this is that the stack is used for temporary storage and that some of the values are register values that are stored in RAM before the device enters the interrupt handler. These values will be popped back before leaving the interrupt handler.

Note: Atmel Studio will by default cache what is in flash and RAM so that not all of flash will be read from the device and updated in the Memory view. To be certain that the values shown in Memory view are updated press "Alt + F7" and after the "Cache all flash memory except" write the memory range of flash that is being written to (in the project code this range is 0x3FF80, 0x3FFC0). The reason to not turn of all caching is that debugging will be very slow as the device will then have to transmit all of flash and RAM memory after each break point.

Figure 2-3. Processor View

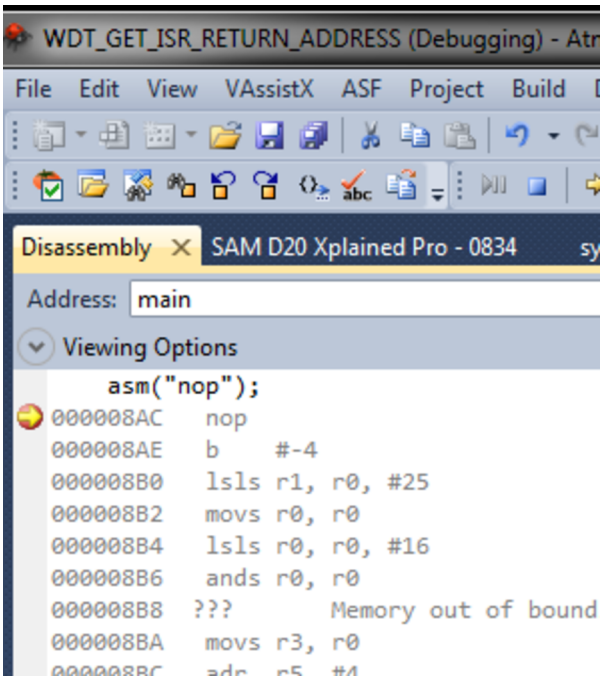


To find the return address it is necessary to know where to look in the stack. To start with we should setup a test where we know what to look for. To find this we can force the interrupt to be fired when we know where the code is executing. The easiest way to do this is to add a while(1) loop in the code that will not let us reset the WDT timer.

Now that it is known where the code will execute when it is interrupted, it is possible to find the address for this execution. The easiest way to do so is to run the code in a debug session, break it and open the Disassembly view in Atmel Studio (Alt + 8).

As can be seen from Figure 2-4 Disassembly in Debug Mode the while loop can be found on address 0x000008AE. Now we can match this with what is in the Memory view. In the Memory view we see that this value can be found in flash address 0x0003FFA0 this is 9 words (a word is 32-bits) below the top of the stack. If we run this code on a project where it is not known where the WDT_Handler returns this is where we can expect to find the return address.

Figure 2-4. Disassembly in Debug Mode



Note: The use of the breakpoint in the code can help find the correct line in disassembly.

Note: How many words below the top of the stack the return address can be found may change with compiler options and the code size of the WDT_Handler() function. Always check this position on the project that is being debugged with the same compiler options and the same interrupt handler.

2.2. Reading Flash

When a device has failed and it is not connected to a debugger it is necessary to read out the flash in order to find out what happened.

To do this connect the device to a debugger and open Atmel Studio. Open the "Device Programing" view (CTRL+SHIFT+p). Select the tool that is used for debugging the device and select the correct device and press "Apply". Now go into the "Memories" menu and under the flash heeding select "Read". A prompt will open up where it is necessary to select a name of the hex file and the folder where to store it. Save the file.

2.2.1. Intel Hex Files

The file can now be opened and inspected. The hex file follows the Intel® hex standard:

```
:llaaaattFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFcc
```

- The colon starts the record
- ll is the length of the record
- aaaa is the last 16-bytes of the address
- tt is the record type
  - 00 - data record
  - 01 - end-of-file record
  - 02 - extended segment address record
  - 04 - extended linear address record
  - 05 - start linear address record (MDK-ARM only)
- F is the data field. Can be any length but in this case 16-bytes
- cc is the one byte checksum of the record

Note: The endianes is different for the data in this file than what is seen in Atmel Studios Memory viewer.

2.3. Precautions

Some precautions must be taken with this debug method.

Writing to flash takes some time and must be factored into how long before the WDT resets the device the early warning interrupt is set to fire. The maximum time of a page write to flash is 2.5ms.

The OSCULP32 is designed for low power, not to be highly accurate. If this oscillator is used for the WDT this must be taken into account for the timing.

If the WDT does a chip reset due to the CPU trying to access a register in a peripheral where the synchronous APB clock is turned off this will cause a bus stall. In this case the CPU will never enter the interrupt handler and flash will never be written to. If this is the case, check the Power Manager registers in a debug session and see if any of the synchronous clocks are turned off that should not be.

In the Atmel Studio project the WDT_Handler writes 15 32-bit values into one flash page. If 16 values had been written the NVM write command can/must be omitted depending on the value of NVM->CTRLB.bit.MANW.

2.4. Determining the Reset Cause

If it is not certain that the device is being reset by the WDT there is a way to check what the reset cause is.

In the device there is a Reset Cause (RCAUSE) register that contains the latest reset source. What peripheral the RCAUSE register is located in may differ from device to device, see the datasheet for the device for more information.

This register shows the last reset cause and for that reason it is not possible to use a debug session in studio to show the last reset cause. This because Atmel Studio will reset the device before entering

debug mode. It is possible to write the reset cause to flash and then read out the hex file without causing a reset.

The register value can be written to flash each time main is entered. Note that bits in flash that are not written contains a one. The register value should therefore be inverted to get as few one-to-zero transition as possible. If this is done then all the ways in which a device has been reset can be stored in one 8-bit value in flash.

Note: Check the cache settings in Atmel Studio before reading values from flash in the Memory view.

3. Revision History

Doc. Rev.	Date	Comments
42248B	04/2016	New template and some minor corrections
42248A	04/2015	Initial document release

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**