

Le71SDKAPIL VoicePath™ API-II Software p2.19.0.2

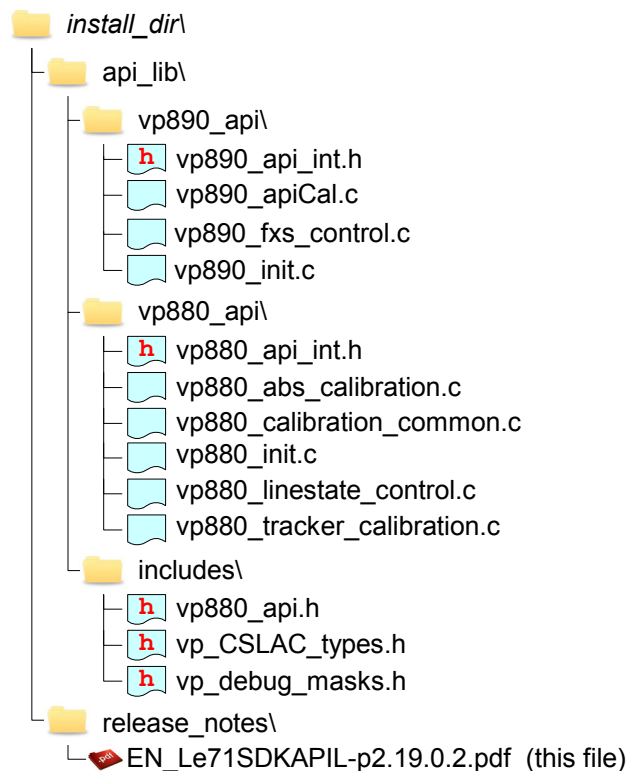
Nov 27, 2012

This document describes the p2.19.0.2 patch release which fixes known issues in the Le71SK0002 VoicePath™ API-II Software Version P2.19.0.

1.0 REVISION SUMMARY

This is a Patch release of the Le71SDKAPIL software package. The content of this release replaces content from P2.19.0.

The patch release contains the following files:



2.0 CORRECTED ERRATA FROM P2.19.0

This section describes errata in P2.19.0 that have been corrected in this release.

VE880 and VE890

- C1 Corrected:** Increase the minimum VAS value.
Impact: VAS margin improvement.
- C2 Change:** Ensure that all mathematical computations in the code do not require more than an int16 format or are promoted to int32 where it is needed. The following chapter details where changes were necessary.
Impact: The API-II C code is now fully compatible with a 16 bit platforms.
- C3 Change:** Ensure that all the API-II debug messages are now compatible with the changes implemented to support 16 bit platforms.
Impact: The API-II debug messaging implementation is now fully compatible with a 16 bit platforms.

VE880

- C4 Corrected:** Prevent *abvError* device object data from being corrupted.
Impact: Corrupted data would be programmed into the Battery Calibration Register if *VpInitLine()* or *VpConfigLine()* were called after *VpInitDevice()*.
- C5 Corrected:** Prevent writes to the profiles loaded in memory.
Impact: Improve compiler compatibility.
- C6 Corrected:** Prevent a temporary signal disruption (4μs) on one channel while the other channel is performing *VpInitLine()*.
Impact: Improve quality of service.

3.0 COMPUTATION ISSUES THAT WOULD IMPACT A 16 BIT PLATFORM

3.1 OVERFLOW ISSUE

C compilers resolve math expressions from the left to the right if all operators have the same precedence (like * and /). For each sub-computations, if the formats are different, the compiler promotes one of the two operands to the larger format. On a 32 bit system, even if the two operands are 16 bit, the math would be done with 32 bits and saved in 32 bit registers. The following example describes what typically happened in the 2.19.0 API.

Wrong: $(int32)result = (int16)A * (int16)B / (int32)C$

The compiler will first multiply *A* and *B* then divide by *C*. As *A* and *B* are both *int16*, the result will be processed as *int16* and saved as *int16*, *C* format is ignored during this step. If the product of *A* times *B* is greater than 2^{15} , the computation would overflow on a 16b system and corrupt the final result.

Right: $(int32)result = (int16)A * (int32)B / (int32)C$

A and *B* have different formats, so the compiler will promote *A* to *int32* then all the rest of the computation will be done in 32 bits whatever the system.

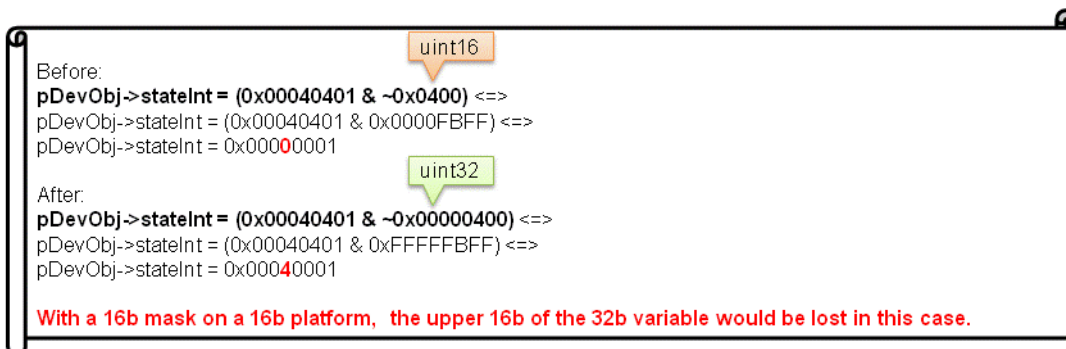
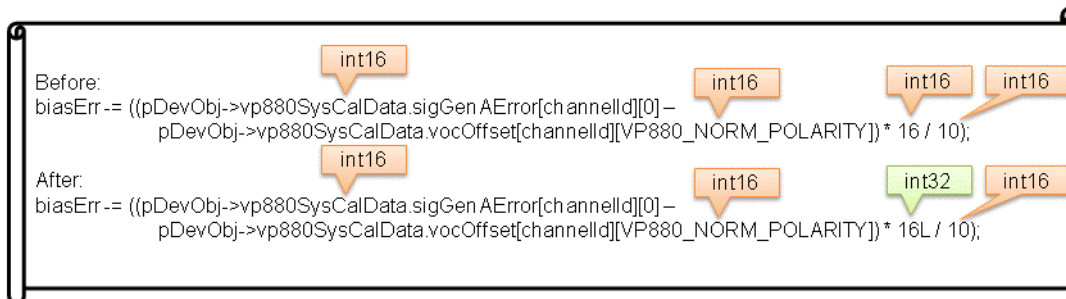
3.2 MASKING ISSUE

Bit masking is commonly used to keep track of internal status. One status variable is declared and several masks are defined for each status. Status variables and masks must have the same format to ensure a correct masking operation.

If the variable is declared as 32 bit and some of the masks are 16 bit, it can cause an issue on a 16 bit platform. This issue doesn't exist on a 32 bit platform, because all the defines are 32 bit by default. An example will illustrate this issue in the section 4.1.

4.0 VP_API-II EXAMPLES

4.1 COMPUTATIONS THAT WERE FIXED TO WORK ON 16 BIT SYSTEMS



4.2

COMPUTATIONS THAT WOULD NOT OVERFLOW ON A 16 BIT PLATFORM

Throughout the code review of the entire API, we identified equations that would not cause any issues on a 16 bit system, because some of the input parameters are limited by design.

API computations that remain unchanged and are correct on a 16 bit system:

The diagram shows a code snippet with variable type annotations. A green box labeled 'int32' points to 'abvTarget'. A pink box labeled 'uint8' points to 'calData.calSet'. Two orange boxes labeled 'int16' point to the '5' constants in the expression. The code is: `abvTarget = (pDevObj->calData.calSet * 5) + 5;`

"pDevObj->calData.calSet" is limited to 255, the maximum is then $255 \times 5 + 5 = 1280 < 2^{15}$
If "pDevObj->calData.calSet" was an int16, it could have been an issue.

The diagram shows a code snippet with variable type annotations. A green box labeled 'int32' points to 'abvTarget'. An orange box labeled 'int16' points to 'coarseVoltTarget'. Two orange boxes labeled 'int16' point to the '5' constants in the expression. The code is: `abvTarget = (coarseVoltTarget * 5) + 5;`

"coarseVoltTarget" is limited to 160V by design, the maximum is then $160 \times 5 + 5 = 805 < 2^{15}$
If "coarseVoltTarget" was not capped to 160 by design, it could have been an issue.

The diagram shows a code snippet with variable type annotations. A green box labeled 'int32' points to 'pcmTargetVoc'. An orange box labeled 'int16' points to 'targetVoc'. Another orange box labeled 'int16' points to the '3' constant in the expression. The code is: `pcmTargetVoc = (int32)(targetVoc * 3);`

"targetVoc" is limited to 57V by design, the maximum is then $57 \times 3 = 171 < 2^{15}$
If "targetVoc" was not capped to 57 by design, it could have been an issue.



Microsemi[®]

Microsemi Corporate Headquarters
One Enterprise, Aliso Viejo CA 92656 USA
Within the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996

Microsemi Corporation (NASDAQ: MSCC) offers a comprehensive portfolio of semiconductor solutions for: aerospace, defense and security; enterprise and communications; and industrial and alternative energy markets. Products include high-performance, high-reliability analog and RF devices, mixed signal and RF integrated circuits, customizable SoCs, FPGAs, and complete subsystems. Microsemi is headquartered in Aliso Viejo, Calif. Learn more at www.microsemi.com.

© 2012 Microsemi Corporation. All rights reserved. Microsemi and the Microsemi logo are trademarks of Microsemi Corporation. All other trademarks and service marks are the property of their respective owners.