Known Bugs in ETEC Version 2.44

Bug Identifier	Source	Problem/Bug Description	Severity	Workaround Description	Affected Releases	Fixed Release
V1.00D-5 (2009-Dec-15)	internal	When the sizeof operator is applied to a constant the wrong size may result, e.g. sizeof(1) may result in "1" rather than the expected "3" bytes.	2	Take the sizeof the desired type instead: sizeof(int)	All versions	TBD
V1.20A-14 (2009-May-20)	internal	Chan interrupt opcodes may be moved relative to adjacent RAM instructions by the optimizer. This may cause unexpected results, particularly in the case of a DMA interrupt.	3	Use _OptimizationBoundaryAll() or #pragma opimization_boundary_all if there is concern that an interrupt may cross a critical RAM access.	All versions	TBD
V1.25A-11 (2009-Sep-28)	internal	If pointer arithmetic generates a negative result, and the object pointed to is larger than 1 byte in size, ETEC code will generate an incorrect result. This is because an unsigned shift (or unsigned divide) is applied after the pointer arithmetic to convert from byte addressing to object indexing.	3	Keep pointer arithmetic results in the non-negative domain.	All versions	TBD
V1.25B-6 (2009-Dec-9)	internal	The _STACK_SIZE_ defines macro gets the calculated value of the worst-case stack depth. In certain rare cases, this value can be slightly larger than the actual worst-case. This can occur when a stack usage of a register save and restore (e.g. in a called C function) is eliminated via optimization. Such a register save requires 4 bytes of stack space, but the removal of it is not currently getting accounted for in the stack size calculation.	4	Care should be taken in that in some rare cases, a _STACK_SIZE_ value that is non-zero can still mean that no stack is actually utilized. Another way to verify that no stack is used is to make sure that no <func/class name>__STACKBASE_ macros are defined.	All versions	TBD

V1.25B-7 (2009-Dec-11)	internal & customer	The optimizer/analyzer does not yet support reentrant functions, whether they be callable C functions or ETEC code fragments. Reentrance is supposed to be detected and cause an error, but in some cases this detection failed, allowing for optimization to continue. Sometimes the result could be a linker crash, or sometimes invalid code generation, or in some cases working code resulted.	3	Avoid writing reentrant functions until the ETEC optimizer/analyzer fully supports them.	All versions	V1.25C (reentrance detection), TBD (support reentrance)
V2.23B-5 (2014-Mar-18)	internal	In some cases when making a fragment call, and the fragment is contiguous with the calling code (i.e. jump can be eliminated), the link-time optimizer mistakenly optimizes out code it should not.	2	This situation, if encountered, can be corrected by re-arranging the code to prevent the fragment call and fragment code from being continguous.	V2.00A and newer	TBD
V2.23B-7 (2014-Jun-6)	customer	The C preprocessor is currently allowing the same macro to be expanded in multiple replacement passes, which causes the preprocessor to break when such "recursion" is encountered.	3	Avoid self-referencing preprocessor macros.	All versions	TBD
V2.44A-1 (2017-May-22)	internal	When an array is accessed via a constant index, AND the array is global and only exposed via an extern declaration, compilation may fail.	3	Access the array via a variable index, rather than constant, even if a new temporary variable is required. Or, rearrange code so that it sees the actual array definition, rather than just the extern declaration.	All versions	V2.50A
V2.44A-2 (2017-May-25)	customer	When using eTPU-C mode (if-else entry table structure) combined with inline assembly and 16-bit channel frame variables, compilation fails.	3	Avoid the combination of 16-bit variables and inline assembly (inline assembly should be avoided always anyways), or package the 16-bit variables in a structure of 3 or more bytes in size.	All versions	V2.50A
V2.44A-3 (2017-Jun-20)	customer	When a testable channel flag or condition code flag (e.g. channel.TDLA or CC.Z) are directly under a conditional expression operator (?: 2nd or 3rd argument), compile/link fails.	3	Use if-else instead of conditional expression in this case.	All versions	V2.50A

V2.44A-4 (2017-Jul-04)	customer	In cases of arrays with elements larger than 0x7F bytes, and where the array is indexed by a signed 8 bit variable / expression, the address calculation code could be generated incorrectly.	2	Typecast non-24-bit variables/expressions used as array indices to unsigned int24.	All versions	V2.50A
V2.44A-5 (2017-Jul-06)	customer	The duplicate expression optimization could fail in the case where the last duplicate expression falls within a loop, and the start was outside this loop, resulting in data corruption.	2	Disable the duplicate expression optimization with the option "-optDis=0x20".	V2.00A and newer	V2.50A

Bug Severity Level Descriptions:

1 – Problem causes complete work stoppage. No work-around is possible. The problem is likely to be hit by most users. This level of bug will typically trigger a new release or patch in a short time frame.

2 – A difficult problem to track down, such as incorrectly generated code. Typically there is a work-around available for this kind of bug.

3 – A bug that is easy to spot, and/or generally has a straight-forward work-around, or has minimal impact.

4 – Not truly a bug (i.e. tool is within spec.), but rather something that might affect compatibility or usability. Work-arounds available.