

Known Bugs in ETEC Version 2.60

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 of 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 optimization_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 contiguous.	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
V2.60C-1 (2018-Nov-7)	internal	If a 'C' variable is referenced in inline assembly, and it is not declared within the very same source file, the inline assembly process will fail to find the variable reference.	3	Add an extern reference to the problem variable in the source file that contains the inline assembly.	All versions	V2.61A
V2.60C-2 (2018-Nov-16)	internal	The compiler was not reporting an error when duplicate eTPU class method/thread definitions exists with the same source file. This would not link however.	4	Eliminate duplicate eTPU class method/thread function definition.	All versions	V2.61A

V2.60C-3 (2019-Feb-15)	internal	Generating a standard entry table via the if-else array can have issues with the 6 non-HSR vectors that have don't care pin values in certain unusual cases.	2	The issue can be worked-around by more thoroughly defining the entry conditions. The follow code snippet comes before any of the lsr==1 cases wherein pin is don't care are allocated, and they are not getting picked up: else if (((m2==1) && (pin==1) && (flag0==0)) ((m2==1) && (flag0==1) && (pin==0))) Adding the following condition gets the missing vectors: else if (((m2==1) && (pin==1) && (flag0==0)) ((m2==1) && (flag0==1) && (pin==0)) ((m2==1) && (lsr==1))) /* work-around entry table issue */))	All versions	V2.61A
V2.60C-4 (2019-Feb-28)	internal	If a structure has an alignment such that the modulo 4 of its size is 3 (unusual; in the last 4-byte word of the structure only the first 3 bytes are used), and an assignment copy from one object of the type to another is done, that last word does not copy correctly.	3	Either avoid the assignment copy or add 1 byte to the structure in order to prevent the problem.	All versions	V2.61A
V2.60C-5 (2019-Feb-28)	internal	If typedefs of array types are used for the type definitions of function parameters, compilation will fail.	3	Use the raw type rather than the typedef to declare the function signature(s).	All versions	V2.61A

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.