Known Bugs in ETEC Version 2.20

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)
V2.20B-1 (2013-Apr-09)	customer	The linker auto-defines feature is outputting multiple definitions for global enumeration declarations in some cases. This occurs when the declaration is included in multiple source files that are compiled into separate object files that are then linked. This problem should not make the auto-defines file unusable, but may lead to warnings in host code compilation.	4	None necessary. The C preprocessing standard is such that re-definitions are allowed as long as the re-definition is an exact match.	V2.20A-B	V2.21A
V2.20B-2 (2013-Jan-20)	customer	The linker reports "cna loop" in the WCTL analysis, even though there is no code loop, in some cases where threads or fragments are called from other functions/threads/fragments. This means the WCTL is unable to automatically calculated for the thread in question.	3	None in general. An optimization boundary placed right after the offending call to a fragment may work around the issue.	All versions	V2.21A
V2.20B-3 (2013-Feb-13)	customer	When the eTPU2 interrupt-from-current-channel feature is used, the optimizer may move it across a channel change boundary which is incorrect. The correct behavior is for a channel register change to block the optimization movement of this sub-instruction.	2	Use an optimization boundary to prevent instruction movement across a channel change boundary.	All versions	V2.21A
V2.20B-4 (2013-Aug-19)	customer	If a parameter to a function call is invalid (e.g. an undeclared symbol), the compiler can crash instead of reporting the compiler error.	3	Fix the non-compilable code.	All versions	V2.21A

V2.20B-5 (2013-Sep-19)	internal	If a cast to void* is used on a constant value, the compiler will lock-up. g_ptr = (void *)0x11; E.g. the above line of code triggers the problem.	3	Do not use a (void *) cast on a constant value.	All versions	V2.21A

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.