Known Bugs in ETEC Version 1.30

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
V1.00D-7 (2008-Dec-15)	internal	It appears that ETEC integer promotion rules are not correct in all cases. For example, the code { unsigned char a = 1; unsigned char b = 2; int c = a-b; } should yield a value of -1 in c but instead ETEC-generated code results in 255.	2	Cases such as the example shown can be corrected through the use of explicit typecasts, e.g. int c = (int)a - (int)b;	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)
V1.30A-1 (2010-Apr-26)	customer	When the source file being compiled is referenced via a path AND it contains inline assembly, the debugging information for that file may come out invalid. This is most likely to occur if the path contains forward slashes. For example, if the following is executed : "ETEC_cc.exe ../src/etpuc_somefunction.c", and etpuc_somefunction.c contains inline assembly, this bug will be encountered.	3	Two different work-arounds. One is to not use a relative or absolute path in the build script or makefile (do the build directly from the source directory). The second is to use backslashes in the path. This may not be possible in some cases, e.g. makefiles do not like backslashes.	V1.25A-V1.30B (pre-V1.25A releases would have had a somewhat different set of pathing issues with inline assembly)	V1.30C

V1.30C-1 (2010-Jun-7)	internal	Enumerator name clashes are not generating a compilation error. E.g. the following code should fail to compile.
		enum ENUM1 { LIT1, LIT2 };
		enum ENUM2 {
		ENUM2_VAL,
		LIT1, // should error due to conflict with previous LIT1 definition
		};
			3	Avoid name conflicts in enumerators. One typical way to do this is to always prepend the enumeration tag name to each enumerator, thereby ensuring a unique name.	All versions	V1.31A

V1.30C-2 (2010-Jun-9)	customer	In a rare case an int8 channel variable located at offset 5,6, or 7 (relative to the CPBA) can get overwritten with a garbage value. It is theoretically possible that there are other instances of this bug, however the compiler code generator does not appear to generate the code sequence that triggers the optimizer bug in any other situation. The optimizer incorrectly uses the diob preload instead of the load to the P register. The P register is required because the ALU cannot access individual bytes in the diob register.	2	Disable optimization on the problem line of code by encapsulating it with optimization boundaries. E.g.
				#pragma optimization_boundary_all
				u8_var = 5;
				#pragma optimization_boundary_all
					All versions	V1.31A

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.