Known Bugs in ETEC Version 1.31

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	V2.00A
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)
V1.31A-1 (2010-Jun-23)	customer	When a function with an argument of enum type is called, if the passed parameter is a constant (including enum literals) the compilation would fail.	3	Change the function argument type to an "int".	All versions	V1.31B

V1.31A-2 (2010-Jul-05)	customer	When a scratchpad programming model is used, there are cases where scratchpad variable and temporary locating may be done wrong, resulting in multiple items using the same location with resulting data corruption. The bug case is generally triggered by 8-bit scratchpad variables, particularly in cases where there are more 8-bit variables than 24-bit, intermixed with the definition of a function that takes parameters (a non-empty parameter frame). Note that the .map file displays all scratchpad variables at the location they are supposed to be at; when the bug occurs a variable (or variables) of stored at a different location than that indicated in the .map file.	2	This bug can generally be worked-around by moving the function definition with the conflicting parameter frame to the very beginning of compilation/link, or the very end, or by reducing the number 8-bit variables in scratchpad memory. For example, _Bool scratchpad variables could be converted to unsigned int24 type. In any case, careful analysis must be done to ensure the bug has been completely worked-around.	V1.25B - V1.31A	V1.31B
V1.31A-3 (2010-Jul-07)	customer	The ETEC linker is issuing an error when mismatching function declarations are encountered. An error should only occur when a declaration and definition of a function mismatch. Since the return type of the absolute value intrinsic functions was changed between V1.25B and V1.25C (return type is unsigned ratehr than signed now), attempting to link object files built with V1.25B and earlier with object files built with V1.25C is incorrectly failing due to the declaration mismatch detected.	3	Ensure that object files being linked are compiled with a coherent set of declarations. With regards to the absolute value intrisnic function mismatch, this may require editing the ETpu_Lib.h or similar to get around the issue.	All versions	V1.31B
V1.31A-4 (2010-Jul-07)	customer	The ETEC linker issues an error and stops the link process when .lst file output is requested but not all underlying source files that contribute to the executable image are available.	3	Do not attempt to generate .lst files when all source files are not available, or use dummy source files to get around the error.	All versions	V1.31B

V1.31B-1 (2010-Jul-15)	internal	The error/warning messages are at times outputting source file path information inconsistently. In some messages styles, the path can be reported incorrectly. Compilation and linking occurs correctly; this is a messaging issue.	4	None needed.	All versions	V1.31C
V1.31B-2 (2010-Aug-06)	internal	If a function contains an intrinsic that uses the MDU or a fixed point library call, but does not use any other MDU operators (multiply/divide), the code generator may mistakenly assign a local variable to the macl (or mach) register which may conflict with the intrinsic or fixed point call.	2	Add a useless line of code to the problem function such as "if (0) dc *= dc;" where dc is a local variable (or any variable for that matter). The optimizer removes all code associated with this dummy line of code, but it is enough to cause the code generator to avoid using the mach/macl registers for local variables.	All versions	V1.31C

V1.31C-1 (2010-Aug-25)	internal	In the case where one operand of a binary operation is of integer type, and the other is of fract type the code generator may incorrectly swap the order of the two operands which is a problem for operations that are not commutative. The problem happens when the second operand is the fract. For example, the following would compile incorrectly:
		unsigned int24 i_am_uint24;
		unsigned fract24 i_am_ufract24;
		// ...
		if (i_am_uint24 < i_am_ufract24)
		{
		// ...
		}
		The operands get swapped so in essence a '>' operation is done.
			3	The problem is easily worked-around by making sure the types of operands are explicitly made the same, such as:
				unsigned int24 i_am_uint24;
				unsigned fract24 i_am_ufract24;
				// ...
				if (i_am_uint24 < (unsigned int24)i_am_ufract24)
				{
				// ...
				}
					All versions	V1.31E

V1.31E-1 (2010-Oct-20)	internal	The expression (x != 0) is supposed to result in a 0 or 1, but is instead resulting in a non-zero value rather than 1 when the expression is true. This is fine when the expression is used in a test such as an if () statement, but a problem when the expression result is used in arithmetic.	2	Change statements such as "result += (x != 0);" to "result += !(x == 0);" or "result += (x ? 1 : 0);".	All versions	V1.31F

V1.31E-2 (2010-Nov-2)	customer	There is an optimization bug when a chan register write is closely followed by a read of an ert register (erta or ertb), wherein the two operations may be incorrectly combined. For example, in this code snippet:
		chan = chan;
		if (erta > ertb) // active time
		{
		The C code above generates assembly code similar to:
		alu chan = chan+0x0;;
		alu sr = ertA+0x0;;
		alu nil = ertB-sr, SampleFlags;;
		seq if LessThan==false then goto ELSE_IFELSE_37, flush;;
		Which optimization improperly combines into:
		alu sr = chan+0x0;;
		alu nil = ertB-sr, SampleFlags;;
		seq if LessThan==false then goto 0xD60, flush;;
			2	The workaround is to create an optimization boundary between the chan register write and the ert register read, like:
				chan = chan;
				#pragma optimization_boundary_all
				if (erta > ertb) // active time
				This results in correct code:
				alu chan = chan+0x0;;
				alu sr = ertA+0x0;;
				alu nil = ertB-sr, SampleFlags;;
				seq if LessThan==false then goto 0xD64, flush;;
					All versions	V1.31F

V1.31E-3 (2010-Dec-2)	internal	Cases have been found where the const type modifier has been getting misapplied to other non-const declarations in the area of a const declaration, which can result in undue compilation failures. The problem may be tied to the use of 'const' in function declarations.	3	Disable use of const when a problem arises for now.	All versions	V1.31F

V1.31E-4 (2010-Dec-2)	internal	When an array that is a member of a structure needs to be converted to a pointer type (e.g. passing a reference to the array into a function call), compilation is failing.	3	If compilation fails due to this bug, rewrite the offending expression. For example: WAS (failing): struct_var.array_member TO: &(struct_var.array_member[0])	All versions	V1.31F
V1.31E-5 (2010-Dec-15)	customer	When using the -enginescratchpad programming model there are cases where the defines file output causes the linker to file, or in some cases the defines file will process but contain corruption in the engine variables area.	2	No reasonable workaround. The short-term method to move forward until the problem is fixed is to disable the defines file output (-defines- link option) or not use the -enginescratchpad programming model.	All versions	V1.31F
V1.31E-6 (2010-Dec-19)	internal	When a no-flushed conditional branch/call is followed by a conditional ASCE, the linker's internal diagnostics falsely detects a bug (this case can only be generated with assembly; the compiler qill not generate this opcode sequence). For example: seq if n then goto MultiCondionalAlu_Neg, no_flush;; alu if n then p = p + 0x32;; Yields the error: Fatal Bug Detected During Internal Diagnostics: An atom is both a fork and a join, DiagCheckAtomMemChain();	3	The assembly must be re-coded to work-around this.	All versions	V2.00A
V1.31F-1 (2011-Apr-11)	customer	When the linker option "-idata-" is specified to disable initialized data files, it has the side-effect of not properly initializing the _GLOBAL_INIT_DATA_ADDR_ macro in the auto-defines file.	3	Do not specify -idata- if the auto-defines macro _GLOBAL_INIT_DATA_ADDR_ is needed.	All versions	V2.00A

V1.31F-2 (2011-May-9)	customer	When the stack programming model is used, and a function call is made that takes at least 3 24-bit parameters, a bug may occur in the passing of the parameter values. It can come about if the parameter variable inside the called function is allocated to a register - the result being that the parameter variable gets the value of a different argument.	2	Re-arranging the function parameters can in many cases work-around the bug, or one of the scratchpad programming models should eb used if possible.	All versions	V2.00A

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.