

## Known Bugs in ETEC Version 2.01

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 <code>_OptimizationBoundaryAll()</code> or <code>#pragma optimization_boundary_all</code> 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 <code>_STACK_SIZE_</code> 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 <code>_STACK_SIZE_</code> 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 <code>&lt;func/class name&gt;_STACKBASE_</code> 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)
V2.01A-1 (2011-Dec-19)	customer	There is a bug when named register variables for p31_0 or p31_24 are used for memory store operations (load is ok). The compiler is incorrectly throwing an error that an invalid register is being used for a load/store operation. The 24-bit p register store (and load) operation works fine via named register variable. The code below results in a compile error due to this bug: <pre> register_p31_0 p31_0;  // ...  some32BitChannelFrameVariable = p31_0; // causes compiler error </pre>	3	Using named register variables is almost like mixing C and assembly. The workaround is to in fact use some inline assembly. Replace the failing code below:  <pre> some32BitChannelFrameVariable = p31_0; *some32BitPtr = p31_0; </pre> With:  <pre> #asm(ram p31_0 -&gt; some32BitChannelFrameVariable.) #asm(ram diob &lt;- some32BitPtr.) #asm(ram p31_0 -&gt; by diob.) </pre>	All versions	V2.10A
V2.01A-2 (2012-Jan-5)	internal	Signed fract operands are being treated as unsigned when doing compares (>, >=, <, <=), for 8, 16 and 24-bit types. This will result in an incorrect result when negative fract values are involved in the operation.	3	The workaround is when comparing signed fract operands, cast them to the signed integer of the appropriate size first. This cast does not generate any code or change the values, but will allow the compare to be generated correctly as a signed compare.	All versions	V2.10A

V2.01A-3 (2012-Jan-17)	customer	When 8-bit data is being written to memory via an array or pointer, it can end up getting written in the first 4 bytes of eTPU memory rather than the intended destination. It is most likely for this to occur when the 8-bit data is being written as the result of a post increment (++) or post-decrement (--) operation. The optimizations of V2.00A made this bug more likely to show, although it has existed all along.	2	The recommended work-around, if it is occurring in conjunction with a post-increment or post-decrement operation, is to change from using that operation. Also, use of temporary variables will likely work around the problem.	All versions (but more likely to occur in V2.00A and newer)	V2.10A
V2.01A-4 (2012-Feb-21)	customer	If the last program flow (jump) in a called function is caused by a 'MAC Spin Loop,' and the function is called multiple times, and every call (seq call instruction) is followed immediately by a thread end (seq end instruction) then the thread can be early-terminated at the 'MAC Spin Loop' such that code after and including the 'MAC Spin Loop' is improperly eliminated.	3	Placing a "#pragma optimization_boundary_all" after the MAC instruction (multiply or divide expression) that triggers the problem provides a low-impact work-around.	All versions	V2.10A
V2.01A-5 (2012-Mar-9)	internal	Found a bug in which an errant error is being falsely detected RAR-Writes are only allowed within an 'RAR Restore Region'. The error occurs when the RAR write moves from inside to outside the region, for instance due to a neighbor joins. Likely only occurs in small two-deep functions.	3	Placing a "#pragma optimization_boundary_all" at the end of the function works around the optimization issue.	All versions	V2.10A

**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.