Known Bugs in ETEC Version 2.10

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)
V2.10B-1 (2012-May-24)	internal	When reading/writing a 32-bit value through a pointer, in a case when the pointer is a complex expression (more than just a symbol), can result in a compilation failure.	3	If necessary, use a temporary int32 pointer variable to perform the read/write, so as to avoid that complex expression that helps trigger the problem.	V2.10A-B	V2.10C
V2.10C-1 (2012-Sep-13)	internal	A structure of only _Bool type members, or that ends with _Bool type members, could fail to compile.	3	Either use a union (with a struct of _Bool members under it, as well as a dummy member of the appropriate size), or use bitfields instead.	All versions	V2.20A
V2.10C-2 (2012-Sep-13)	internal	The ability to disable autostruct file generation is not working. (-autostruct- linker option)	2	None; ignore generated file.	V2.10A-C	V2.20A
V2.10C-3 (2012-Sep-26)	customer	Array indexing via an index variable generates incorrect code in the following case : the index variable is of type int8/char (signed), and the array element size is greater or equal to 128 bytes.	3	Use an index variable that is unsigned or bigger than 8 bits in size.	All versions	V2.20A

V2.10C-4 (2012-Nov-13)	customer	In certain program flow situations that form a 'Fork' (e.g. if/else) where the first instructions along all paths are matching and parallelizable channel instructions [e.g. ClearMatchALatch();, ClearMatchBLatch();] and the channel instructions along ALL paths match, except that the earlier segment(s) contains more parallelizable channel sub-instructions than the last segment that fit in the same opcode, then it is possible that the additional sub instructions in the first segment that are not found in the last segment can get incorrectly eliminated. For example, the following code is affected by this optimization bug - the ClearMatchBLatch() sub-instruction gets removed erroneously:
		
		if (IsMatchBLatched())
		{
		  ClearMatchALatch();
		  ClearMatchBLatch();
		  someVar = 0x23;
		}
		else
		{
		  ClearMatchALatch();
		  someVar = 0x77;
		}
			2	One work-around is to protect the set of channel sub-instructions that can get packed into a single opcode with an explicit atomic region - this can be done as follows:
				if (IsMatchBLatched())
				{
				#pragma atomic_begin;
				  ClearMatchALatch();
				  ClearMatchBLatch();
				#pragma atomic_end;
				  someVar = 0x23;
				}
				else
				{
				  ClearMatchALatch();
				  someVar = 0x77;
				}
				
				Or, the code can be modified slightly to avoid the problem:
				- reversing the ClearMatchALatch() and the ClearMatchBLatch() lines in the if-clause above avoids the issue.
				- bringing the common ClearMatchALatch() out before the if statement also avoids the issue.
					V2.00A - V2.10C	V2.20A

V2.10C-5 (2012-Dec-20)	internal	A bug can occur in a switch statement under a fairly unusual set of circumstances. If the first statement in a 'case:' is a read from data ram that uses the DIOB register, and the last use of the diob register prior to the 'switch' was also a read of this same data ram location, the bug may occur. The bug is that the data ram read in the case statement can get incorrectly eliminated.	3	Place an optimization boundary just after "case:". E.g., add the line "#pragma optimization_boundary_all".	All versions	V2.20A

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.