Known Bugs in ETEC Version 1.11 – 1.12

Bug Identifier	Source	Problem/Bug Description	Severity	Workaround Description	Affected Releases	Fixed Release
V1.00D-5	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 "3".	2	Take the sizeof the desired type instead: sizeof(int)	All versions	TBD
V1.00D-7	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
V1.10A-3	customer	If a channel frame (or structure) contains a 16-bit piece of data, and two 8-bit pieces of data, packed such that the 16-bit data is at the start of a double-even address, followed by the two 8-bit items (thus making up an entire 4-byte word), then read access of the 8-bit data fails due to invalid code generation.	2	Change the data types or re-arrange the channel frame to avoid the occurrence of such data packing.	All versions	V1.20A
V1.11A-1	internal	When a structure(union) contains a member that is itself a structure(union), if the member has been declared with a typedef of the structure(union) then referencing this structure member results in a compilation error.	3	The workaround is to declare structure/union members of type struct/union using their base type rather than a typedef.	All versions	V1.20A
V1.11A-2	customer	Enumeration literals are not getting replaced in #pragma write directives. For example, #pragma write h, (#define ENUM_VAL_0 EnumVal0); where EnumVal0 has a value of 5 is output as #define ENUM_VAL_0 EnumVal0 rather than #define ENUM_VAL_0 5.	3	One workaround is when defining the enumeration to give its members explicit values (perhaps via macros), which can then be used in the #pragma write.	V1.11A	V1.20A

V1.11A-3	internal	ETEC_cpp.exe is only supporting 7-bit ascii characters, and can crash when 8-bit ascii characters are encountered.	3	Eliminate 8-bit ascii characters from the source code.	V1.10A - V1.11A	V1.20A
V1.11A-4	internal	In some cases, switch statements can fail to jump to the correct code when the value passed to the switch is higher than any of the cases in the switch.	2	If the problem is encountered add a case statement with the highest possible switch value, and have it drop into the same code as the default if it exists, or just break out of the switch statement.	V1.00C - V1.11A	V1.20A
V1.11A-5	internal	The erta register is not being seen as a dependency of a UDCM (eTPU2 user-defined channel mode) write by the optimizer, which could result UDCM getting written with an incorrect value.	2	The synchronization capability _SynchBoundaryAll() or #pragma synch_boundary_all should be used to prevent invalid optimization from occurring.	all up to V1.11A	V1.12A
V1.12B-1	customer	In the C preprocessing stage, specified relative paths are being applied to the current source file path, rather than the original working directory of the compilation, resulting in incorrect source searches.	3	Use an absolute search path instead of relative in such cases.	V1.10A - V1.12C	V1.20A
V1.12B-2	customer	When a MAC register (mach, macl) variable alias is created (via syntax like "register_mach mach;") in an inner scope, and an MDU operation is specified in an outer scope, or previous scope, the MAC register is being seen as allocated and a spurious out of temporary registers error is being thrown.	2	Move the "register_mach mach;" (or similar) declaration to the outermost scope of the function, or even to the global/file scope.	all up to V1.12C	V1.20A
V1.12D-1	internal	When updating the channel flags directly from the p_31_24 register (a seldom used eTPU feature) upstream writes to the p register may be incorrectly eliminated or modified.	2	The code that loads the flags into the p register (p31_24), must be protected with an atomic region; place an _AtomicBegin(); and _AtomicEnd(); around it. This prevents the optimizer from removing it.	All versions	V1.20A

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.