Known Bugs in ETEC Version 1.25

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-1 (2009-Aug-24)	customer	The initialized data files (*_idata.c and *_idata.h are including global variables aliased to register_chan_base along with all the eTPU classes and eTPU functions. This is harmless, but they don't belong there.	4	No work-around necessary.	All versions	V1.25B

V1.25A-2 (2009-Aug-24)	customer	When a channel frame variable is of type struct or union, and its type is specified via a typedef rather than directly via the tag type, then the channel frame variable offset is not output in the auto-defines file (*_defines.h).	3	If the offset is needed in the auto-defines file, then the declaration of the channel frame variable should use the tag type directly to ensure that an offset is generated.
				
				typedef struct phase Phase;
				
				Phase P_A; // no offset generated
				struct phase P_B; // offset generated
					All versions	V1.25B

V1.25A-3 (2009-Sep-1)	customer	When using the scratchpad programming model, the calculated global data size (or engine data size if using -enginescratchpad) is incorrect. The _GLOBAL_VAR_SIZE_ is including some scratchpad data, and thus is greater than or equal to the correct size. While _GLOBAL_SCRATCHPAD_SIZE_ is correct, _GLOBAL_DATA_SIZE_ is not as it incorporates _GLOBAL_VAR_SIZE_. The end result is that user host interface code keying off of these macros may waste some memory.	2	Examination of code, listing and map files can reveal the exact global data usage, but this is a manual process.	V1.25A	V1.25B
V1.25A-4 (2009-Sep-2)	internal	When a code fragment is called from a callable C function (not a thread), link fails.	2	No workaround - such calls cannot be made with V1.25A.	All versions	V1.25B
V1.25A-5 (2009-Sep-4)	customer	When scratchpad (local) _Bool variables are used, they are being tested assuming that the variable is the only _Bool occupying the 8-bit unit, but being updated in bit 0 (LSB) only. This means that is scratchpad memory is not cleared before use, the _Bool variable may not behave correctly. The write of the scratchpad _Bool should be to the whole 8-bit unit.	3	Clear scratchpad memory before allowing eTPU code to run (if it contains local _Bool variables).	All versions	V1.25B
V1.25A-6 (2009-Sep-6)	customer	In some cases, when more code follows a call to a fragment in a function, the object file can be corrupted and the link can fail.	2	No well-defined work-around. Instead of using fragments when this occurs, a regular C function can be called that ends with an _ExitThread() rather than a return.	All versions	V1.25B
V1.25A-7 (2009-Sep-6)	customer	The compiler is outputting a warning message that code is being stranded by a call to a fragment (_eTPU_fragment) when this is not actually the case. This can happen in switch statements, or if such a call is followed by a return statement of a void function.	3	Ignore warning message.	All versions	V1.25B

V1.25A-8 (2009-Sep-8)	customer	Channel frame and engine-relative _Bool type variables have their bit offset macro misnamed in the *_defines.h file that is auto-generated. These bit offset macros are supposed to contain "BOOLBITOFFSET" in the name. The bit offset macros of global _Bool variables, and struct/union _Bool members, do not have the problem. Additionally, the _Bool bit offset macro name for channel frame-related _Bool variables/members is not including the eTPU class/function name.	3	Temporary macros can be used to define the wrong name to the correct name in the host interface code or simulation scripts. Once the correct macro names are being generated in the _defines.h file, the the re-definition macros can be removed and the code will work as-is. For example, if a channel frame variable is defined as: _Bool _b1; With version 1.25A the macro name for its bit offset is generated incorrectly as: #define _CPBA8__b1_ 0x07 The following macro can be defined temporarily to re-name the macro correctly: #define _CPBA8_BOOLBITOFFSET_Test__b1_ _CPBA8__b1_	V1.25A	V1.25B
V1.25A-9 (2009-Sep-14)	customer	When a channel frame contains zero channel variables, and does not require a stack, the channel frame size should be 0. The ETEC linker is sizing channel frames at a minimum of 8 bytes, even in this case when it should be 0 (e.g. the frame size macro in the *_defines.h file gets a value of 8 rather than 0).	3	No real work-around, but also of very little impact (waste of 8 bytes per channel running the problem function).	All versions	V1.25B
V1.25A-10 (2009-Sep-15)	customer	The C preprocessor ETEC_cpp.exe is failing to process directives like #if(TEST == 7) where there is no white space between the "if" directive keyword and the constant expression. As long as the start of the expression is a new token, such as a '(', the preprocessor should handle it without failure.	3	Add a space after the #if.	All versions	V1.25B

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
V1.25A-12 (2009-Oct-7)	customer	In the case of back-to-back function calls, it is possible that an opcode at the end of the first function can get associated with the beginning line of the next function, resulting in a strange .lst file (and a strange mixed source-assembly view in the Simulator) and source line stepping issues in the Simulator. The underlying code behaves correctly.	4	No work-around - ignore the stepping issue in the Simulator. To get all the dis-assembly code at the point of the problem, the Simulator Memory Tool can be used.	All versions	V1.25B
V1.25B-1 (2009-Oct-20)	internal	The compiler is outputting warning message 402 that unsigned mutliplication or division is being done even though one operand is signed in some cases when it should probably be suppressed. The case is when the signed operand is a constant - by default constants are of signed type. A note on signed vs. unsigned multiplication in the eTPU: the 24-bit result in the MACL register is the same regardless of which multiplication type is done, however, the upper 24-bit MACH result may differ, as well as the state of the MN condition code flag.	4	The constant operand can be typecast to unsigned (e.g. change "10" to "(unsigned int24)10"), or the warning can just be ignored as there has been no change to how the actual code is generated.	V1.25B	V1.25C
V1.25B-2 (2009-Nov-11)	customer	The analysis file (*_and.html) output by the linker had a problem wherein the error handling library was not being listed in the Source Code Information section, ands its opcodes were mistakenly being added to the last source file in memory, thereby providing misleading information.	4	Look at the .map file instead - it properly breaks down the code contributions of all source files (translation units).	All versions	V1.25C

V1.25B-3 (2009-Nov-11)	internal	Under certain conditions the linker could fail to link when it shouldn't - specifically in unusual cases where a channel sub-instructions gets joined with a return sub-instruction.	3	Bad code is not generated; rather link fails. The work-around is to place an optimization boundary #pragma at the bottom (or top as necessary) of the function to prevent link failure.	All versions	V1.25C
V1.25B-4 (2009-Nov-20)	internal	Under certain conditions, a const or volatile type qualifier on a basic type can end up getting applied to other symbol references in the linker, which in turn can sometimes trigger a link failure when function prototypes no longer match due to qualifier differences. This is most likely to occur when multiple object files are being linked together, as compared to the include into one source file approach.	2	If a linker type defintion mismatch error occurs when it is expected that there should not be a problem, a const or volatile type qualifier could be causing the problem. If possible, eliminate the qualifier from the source code to work around the failure. A more drastic fallback would be to compile all source as one collective, included source, as this tends to avoid the multiple function prototype issue.	All versions	V1.25C
V1.25B-5 (2009-Dec-9)	customer	When a named register variable is declared via a typedef, and is given a type other than the default register type, the variable is not getting correctly assigned to the named register. // bug: not getting assigned to mach unsigned register_mach mach;	3	Such a variable should be declared directly using the named register feature, rather than through a typedef. // assignment of unsigned int variable 'mach' to register _MACH unsigned register _MACH mach;	All versions	V1.25C
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)

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.