Known Bugs in ETEC Version 1.20

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 the expected "3" bytes.	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
V1.20A-1	customer	When inline assembly is used to create function calls within inline assembly, the function call boundaries must be delineated in order for the optimizer/linker to properly deal with the code. However, at the current time the #pragmas necessary to do this are not supported in inline assembly.	3	Re-write the code to avoid this - use real functions, or in some cases switching to the ETEC eTPU class paradigm can help avoid the issue.	All versions	V1.20C
V1.20A-2	customer	When using ETEC mode, when the definition of a class function or entry table is found, but the function/table if not declared in the class, the compiler/linker does not issue an error until much further down the processing pipeline, making it hard to debug. Error dtection and reporting must be improved for this type of case.	4	None needed.	All versions	V1.20C
V1.20A-3	internal	The 'ram #0' sub-instruction to zero ram does not currently allow symbolic references for the address.	3	Use 'prm<addr>' rather then a symbol name to reference the proper address to be zeroed.	All versions	V1.20C

V1.20A-4	internal	Loading an address constant from a symbol requires use of the '&' token before the symbol name, which does not match existing assembly syntaxes. The requirement for the '&' token needs to be removed.	4	Use the '&' token for now.	All versions	V1.20C
V1.20A-5	internal	When an assembly jump or call dispatch sub-instruction is used with the 'seq' keyword, the assembly process fails. The 'seq' keyword should be optional. [Note: dispatch sub-instructions are of the form 'jump pc+p31_24, flush.' or similar.]	3	Make sure the 'seq' keyword precedes dispatch sub-instructions.	All versions	V1.20C
V1.20A-6	customer	The MAC/MDU intrinsic functions (__macs, __macu, __fmults8, __fmults16, __fmultu8, __fmultu16) do not incorporate a spin loop that waits for the operation to complete, potentially introducing optimization errors. Going forward these intrinsic will incorporate a spin loop. Note that the optimizer will potentially move non-dependent code between the MAC/MDU operation and if possible, eliminate the spin loop.	3	Always follow MAC/MDU intrinsic functions with a check-for-busy spin loop ("while (CC.MB) ;"). If users want full control of MAC/MDU and parallelization, inline assembly can be used instead of an intrinsic function.	All versions	V1.20C
V1.20A-7	customer	The auto defines file is not providing the size of tag types (struct, union, enum). Additionally, anonymous type listings in the auto defines file are not inheriting their typedef name, if it exists.	3	The size of tag types must be found through manual means and hardcoded in host interface code for the time being.	All versions	V1.20C
V1.20A-8	customer	The testing of _Bool type variables against constants using '==' and '!=' operators is generating sub-optimal (but functional) code. Additionally, _Bool variables under logical operations (&&, ||) are generating sub-optimal code (no work-around).	3	Convert tests such as (BoolVar == 0) to (!BoolVar), or (BoolVar == 1) to (BoolVar) - the compiler is properly optimizing these expressions.	All versions	V1.20C

V1.20A-9	customer	When code accesses an extern array variable, the stride size may not be computed correctly.	2	Instead of making an access directly, like "x = extern_array[i];", access it via pointer arithmetic: "x = *(extern_array + i);"	All versions	V1.20C
V1.20A-10	customer	In the ETEC C preprocessor, when running in ETPUC mode, #asm passes through as is even though if found in a function-like macro replacement list it should per the standard generate an error. However, the matching #endasm is not being treated in the same way. Additionally, the ETEC C compiler is only recognizing the #asm, #endasm, and #asm() directives when they are the first non-white space characters on a line.	3	For V1.20A-B, macros must be written in such a way that "#asm" are the first non-white space characters on a source line. Also, when a series of inline assembly instuctions are concatenated in a macro, they should all use the #asm() format for now. When fixed in the V1.20C release, #asm, #endasm, and #asm() will no longer need to be the first non-white space characters on a line to be recognized.	All versions	V1.20C
V1.20A-11	customer	In the auto defines file, the bit offset from the most significant bit of the _Bool unit to the actual bit location is output in a macro containing the BOOLBITOFFSET keyword. It has been found that these offsets are actually too big by 1.	3	As a temporary measure, when the BOOLBITOFFSET macro variables are used they can be adjusted down by 1 (subtract 1 from them).	All versions	V1.20C
V1.20A-12	internal	Function-like macros are failing to be replaced in the text following a #pragma directive.	3	Avoid the use of function-like macros in #pragma text.	V1.20A-B	V1.20C
V1.20A-13	customer	Channel instructions may get illegally re-ordered when atomic (see _AtomicBegin(), _AtomicEnd()) Note that several of the provided channel macros are atomic (see ETpu_Std.h).	2	Use _OptimizationBoundaryAll() or #pragma opimization_boundary_all to prevent chan sub-instructions from moving across boundaries they should not.	All versions	V1.20C
V1.20A-14	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

V1.20A-15	internal	In the past users could access the built-in error handler from user code by generating a function prototype for the error handler label (_Error_handler_entry). One problem with this approach is that the compiler generates a call opcode rather than the more correct jump opcode. The standard error handler entry points will be added to the eTpu_Lib.h header file and given a "return" type of _eTPU_thread which causes the compiler to generate a jump rather than a call. These new prototypes may cause conflicts with existing code that used the old technique.	3	None needed.	All versions	V1.20C
V1.20C-1 (2009-Jun-1)	internal	If a label is placed at the very end of a C function that has a return value, the compiler can crash (note that legal C syntax at least requires a ';' after the label, but that does not fix the crash).	3	Either eliminate the label if it is not needed, or place it inside inline assembly to work-around the problem : #asm(SomeLabel:)	All versions	V1.25A
V1.20C-2 (2009-Jun-3)	customer	When using ETEC mode, the compiler is not in the correct state after processing an entry table defintion (done via the DEFINE_ENTRY_TABLE macro). Depending upon what constructs come after an entry table definition, either bad code can be generated, or with the new enhanced _eTPU_class error checking an invalid error can be thrown, or in many cases the bug is masked and no problems occur.	2	There are two work-arounds available for this problem. One is to make sure that every entry table definition is the very last construct in a translation unit. Note that this is very difficult to do when all code is included into one c file for compilation (i.e. one big translation unit). In this case, the second work-around is recommended. If the entry table definition is directly followed by an eTPU thread definition (e.g. _eTPU_thread Class::Thread(_eTPU_matches_enabled) { /* ... */ }), the processing of the thread definition resets the compiler state and any problematic code generation is avoided.	All versions	V1.25A

V1.20C-3 (2009-Jun-3)	customer	When structs/unions are larger than 4 bytes, the ETEC auto defines file is reporting the internal size of the structure rather than the "stride size" which may include extra padding. For example, "struct S1 { int x; int y; };" is reporting a size of 7 rather than 8 (#define _GLOB_TAG_TYPE_SIZE_S1_ 0x07). Note: the sizeof() operator within eTPU C code is working properly (it would return 8).	3	The size macro can be corrected with some extra code that rounds up to the nearest 4-byte boundary : ((_GLOB_TAG_TYPE_SIZE_S1_ + 3) & ~3). Such code will remain correct even once this bug is fixed.	All versions	V1.25A
V1.20C-4 (2009-Jun-15)	internal	When a switch statement's sub-statement is not a compound statement, compilation is failing.	3	Always make the switch sub-statement a compund statement. E.g. switch (var) { /* … */ }	V1.20C	V1.25A
V1.20C-5 (2009-Jun-23)	internal	A for loop of the following syntax is compiling incorrectly: for (x = 0; x < 4; x++) if (var[x]) { boolvar = 1; /* … */ } The problem lies in having the if statement directly be the for loop's sub-statement.	2	Place the 'if' statement inside { } compound statement markers.	All versions	V1.25A
V1.20C-6 (2009-Jun-29)	internal	_Bool variables that are stored in registers do not get read properly when they are the right-hand expression in a _Bool assignment. E.g. _globalBoolVar = _registerBoolVar fails to generate the proper code, assuming _globalBoolVar is global and _registerBoolVar is stored in a register (e.g. sr).	2	Make sure _Bool variables are not stored in registers if they are used in such a way that the bug manifests itself. In such cases, one work-around is to make the variable static so that it gets stored as part of global memory.	All versions	V1.25A
V1.20C-7 (2009-Jun-29)	customer	When an _eTPU_thread function is the target of a function call (generates a "goto" rather than a "call" under the hood), the optimizer can mistakenly remove opcodes upstream from the _eTPU_thread function that it thinks are uneeded, yet actually are required, resulting in buggy code.	2		All versions	V1.25A

V1.20C-8 (2009-Jul-8)	customer	When two non-constant expressions of type _Bool are compared against each other with either equality or relational operators, invalid code is generated. Such a compare also results in a spurious warning regarding an assignment, even though there is no assignment in the expression.	2	The bug can be worked around by converting the equality/relational operation expression to a series of logical operations; see example below.
				
				_Bool b1;
				_Bool b[16];
				
				// ...
				
				// below C code results in invalid machine code
				if (b1 == b[i]) // ...
				
				// the below C code works around the problem
				if ((b1 && b[i]) || (!b1 && !b[i])) // ...
					All versions	V1.25A

V1.20C-9 (2009-Jul-13)	customer	When a variable of type specified below is declared and used under the conditions stated below, the compiler is falsely throwing a diagnostics error "A 32-bit access must be double even (0, 4, 8, ...)", causing compilation to fail. - array of chars (8-bit type) of length 2, or struct of 2 8-bit types - variable is in a channel (function) frame (i.e. not a global or local variable) - if an array, the error occurs if element 1 is read, e.g. if (SomeArray[1] == 7)... - if a struct, the error occurs if the 2nd member is read, e.g. if (SomeStruct.b == 7) ...	3	The work-around if the problem is an array, is to increase the array length from 2 to 3 (or perhaps better, 4, as it may result in better overall packing of the channel frame). If the problem is triggered by a structure, then the struct type needs to have a dummy member added of size 8-bits (or 16-bits).	All versions	V1.25A

V1.20C-10 (2009-Jun-29)	internal	When a reference is made through a pointer that is a member of a structure, the code generated incorrectly adds an offset to the pointer resulting in incorrect operation.	2	No good work-around exists - if possible, move the pointer out of the structure.	All versions	V1.25A

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.