

SpecC Language Reference Manual

V1.0

Release Notes

The “*SpecC Language Reference Manual V1.0*” technically is based on the contents of the “*SpecC Language Reference Manual*” published as UCI Technical Report ICS-TR-98-13. However, it is authored and copyrighted by Rainer Dömer, Andreas Gerstlauer, and Daniel Gajski. It is the intend of the authors to donate this document to STOC for standardization.

The attached document is final. The following is a complete list of items that, compared to the base document from UCI, have been changed, improved and/or extended:

(1) Clarifications:

- (i.e. issues that the initial LRM left open, that are now clearly defined)
- (1a) type conversion and promotion has been explained in more detail; in particular, the added SpecC types `bit[]`, `bool`, `long long`, and `long double` are converted and promoted in the same manner as the standard ANSI-C types;
- (1b) the types `long long` and `long double` (which are not part of standard ANSI-C, but are supported by the GNU compiler) have been explicitly defined; (in order for non-GNU compilers to be usable, the SpecC frontend will insert required replacements, i.e. `bit[63:0]` for `long long`, into the generated C++ code)
- (1c) assignment of whole arrays (as supported by the GNU compiler, i.e. `int A[10], B[10]; A = B;` is supported in SpecC, i.e. the whole array is copied; (for non-GNU C++ compilers, the SpecC frontend will insert the required `memcpy()` operations)
- (1d) all static variables are initialized with 0 in SpecC, unless explicitly initialized otherwise by the user (in ANSI-C, these would be

- uninitialized); (the SpecC frontend will insert all implicit and explicit initializers into the generated C++ code)
- (1e) concatenation is noted as @, not as & (the latter was a typo in the old LRM)
 - (1f) a minimal set of guarantees about the execution semantics of SpecC programs has been added to specific constructs, such as `wait/notify`, `par`, `try-trap-interrupt`; these execution semantics clearly specify what guarantees are provided by the SpecC language and what is implementation dependent (i.e. depends on the simulation engine being used);
 - (1g) constants can be used in port mappings for `in` ports; in addition, `out` ports can be left open in port mappings; (the user is no longer required to create dummy variables and dummy mappings for these cases)
 - (1h) `bit[8]` is a valid short-cut for `bit[7:0]`

(2) SpecC LRM Fixes:

(i.e. issues that used to have problems or limitations in the old LRM)

- (2a) the `delta` keyword has been taken out; (it was unneeded and only confusing)
- (2b) the `import` statement has been made independent from the SIR file format; instead, any implementation can choose whatever file format for the imported description (i.e. the reference compiler will use source code, the UCI compiler will use binary SIR files, VisualSpec may choose to use XML files; for compatibility among different compilers, every compiler should support source code as well); advantages of `import` over `#include` are: automatic avoiding of multiple inclusion, faster execution (no semantic check required for precompiled files), visible to compiler (imported blocks can be treated special by tools, i.e. IPs)
- (2c) the `pipe` statement is extended for an optional exit condition that will flush the pipeline and terminate the construct, i.e. `pipe(i=0; i<10; i++) { ... }` will execute 10 iterations of the pipeline;

`pipe(; ;) { ... }` is the same as `pipe{ ... }`, whereas the latter one is preserved for backward compatibility; (with the old `pipe` statement, hierarchical composition was not possible)

(3) SpecC LRM Format:

(i.e. textual and formatting issues)

- (3a) extended and improved explanation of the constructs, their semantics and applying rules; (i.e. more text)
- (3b) improved formatting and improved document structure (i.e. numbered rules, improved hierarchy)
- (3c) use of Backus-Naur-Form (BNF) instead of lex/yacc notation (making the LRM independent from an implementation)
- (3d) improved examples with more detailed explanation

Rainer Doemer, March 2001.