

Firmware Development Methodology

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
	v000.1		V

Contents

Development cycle:

1. Create specification of target processor and peripheral hardware if needed.
2. Design architecture of developed software (data structures, partitioning to the modules, specify external and internal interfaces).
3. Implement prototype of software in high-level language (e.g. C). Simulate environment (hardware, external world).
4. Design and develop testing environment (prototype of controlling software, testing engine & framework, nightly/weekly testing).
5. Design test plan and develop test suite for software prototype.
6. Make tests passed on software prototype.
7. Establish low-level programming conventions. Develop environment (or derive it from existing) for low-level programming.
8. Develop instruction set and peripheral simulator.
9. Implement software in low-level basing on high-level prototype. When possible, architecture of software prototype, identifiers, algorithms should be re-used in low-level implementation.
10. Develop and pass module-level tests for low-level implementation. These tests cover independent functions or modules to verify their functionality. Also, some specific or hard to reproduce conditions should be verified in module-level tests.
11. Run testsuite (designed on stage 5) on low-level implementation using instruction set/peripheral simulator.
12. Design and implement code optimizer. Code, produced from assembler sources, accompanied additional information, may be improved automatically to execute faster. Examples of such improvements are:
 - fill the delay slots (if target architecture support delay slots)
 - resolve operand access delays (e.g. destination operand of some command used as source operand of the next command)
 - resolve hazards
 - remove threaded jumps (jump to jump to jump...) and jump to the next command
 - remove void code
 - reorder (invert) branches (using profiling information from previous simulations on prepared test sets) to avoid unnecessary pipeline flushes
 - advanced optimizations, like register renaming to reduce number of memory accesses.
13. Conduct formal code inspection of low-level code. Questionary (inspection check list) should be answered to protect from not detectable and hard to reproduce errors (e.g. interrupt masking, deadlock conditions, register clobbering etc.).
14. (Optional) Interface the testing framework with RTL-level hardware simulator and run tests (or, at least, some of them, because hardware simulation significantly slower than instruction set simulation).
15. Interface the testing framework with silicon hardware and run test suite.

Many of these steps may run concurrently.
