
Firmware

VVV <vvv@oktetlabs.ru>

OKTET Labs proposes a methodology of embedded software/firmware development. We assume that some of following conditions take place:

- software targeted to custom processor or execution unit with limited capabilities (command set, registers, resources)
- it is not reasonable to develop high level language compiler due to limited applications of target processor and its limitations
- target hardware has application-specific units, accelerators, etc.
- size of code quite small (#64-96K essential commands - commands, written by engineer, not generated like unrolled loops etc.)
- strong performance requirements; performance should be measured
- manageable quality of software
- software development in absence of target hardware (e.g. chip specified and its design in progress).

Examples of such applications includes:

- communication devices (ATM controllers, Level 3 accelerators, traffic processing units in network devices, cryptographic processing)
- mass storage subsystem (SCSI host adapters, SCSI devices firmware, mass storage devices, RAID)
- link layers in mobile terminals (GSM, etc)
- graphic accelerators software

In general, developed software intended to perform intensive speed data processing using quite simple algorithms and provides management interface. Non-realtime and complicated processing (e.g. network management, higher-level network protocols) not intended to be implementing in such way.

The following principles are used in our methodology:

- Limit sources of the problem on every stage

```
If tested software compiled using not working compilers and running
on not working hardware or simulator, using incorrect test suite, finding
the source of problem may be quite complex, and many developers should be
involved in this process simultaneously.
```

- Different developers should design code and tests (its quite traditional)
- Common language.

```
When possible, the same, flexible enough, language should be used for code
```

development, testing, tools.

- **Level of assembler language programming should be improved when possible.**

Assembler should support data structure definitions and handling of structured data, function definition/invocation, parameter passing. Assembler should allow assembly-time checks, explicit or implicit (e.g. register usage). Assembler should support structured programming constructs (conditional statements, loops, switches, etc). Function inlining should be supported. Actually, assembler sources written as a program (in prepared environment); result of its execution is executable code.

- **Linking of assembler and execution environment.**

It should be possible to insert run-time checks and trace prints in assembler program. More general, it should be made possible to insert arbitrary high-level code in assembler program intended to be executed when control reaches appropriate place. Of course, it is better to use the same language as used for assembler programming and have access to the context defined in appropriate place of assembler code (function arguments, local variables, etc).

This is quite useful for test development, protective coding, debugging.