

Selecting an ARM Development Environment to Meet Your Project Requirements

A Good Tool is Half the Work

One of the success factors in software engineering is the availability of tools to support software



developers. For some microcontroller families, there are only a few suitable tools, and development engineers are grateful if these tools are at least reasonably stable.

Things are different with microcontroller families (architectures) that have established themselves in the market, such as 8051, C166 or ARM for which a wide range of tools is available. This is actually the perfect basis for successful software engineering - provided it is used efficiently.

Everyone will agree that a C compiler is indispensable. The decision on using an RTOS, however, is often difficult to make, because the related advantages and disadvantages are not always clear. And what about OOP? Should software for new products rather be programmed in C++? Too many times, such decisions

are postponed from one project to the next because there is not enough time for research, and software engineering falls by the wayside.

To meet the demands on both budget and efficiency, it is useful to deal with the requirements and select appropriate tools. For this reason, this paper presents four exemplary development environments.

In all cases, project requirements are given the highest priority. In particular, this refers to requirements like product lifetime, maintainability and extendability as well as the safety and robustness of the software to be developed. Another issue is the level of experience among the software development engineers, the required training times for new techniques as well as the related time-to-market situation.

The tools we have combined to form complete solutions are perfectly synchronized and have already proven successful in many projects. One major advantage is that the development environments can be extended to meet increasing requirements without causing any compatibility problems. This protects the return on investment for tools and training over a long period of time even if engineering requirements change. Extendability is moreover the perfect basis for a gradual introduction of new technologies and thus the distribution of investments and training times over a longer period of time.

Variant 1

This variant is based on the following customer-specific requirements:

Hardware architecture: A single chip architecture is planned for budget and EMC reasons (no external memory).

Real-time requirements: High, however, with only few concurrencies.

Product lifetime: 15 years, no major modifications expected. Maintainability shall be guaranteed over the whole lifecycle.

Time-to-market: Only a few months are available for implementation **Time To Market:** es stehen nur wenige Monate zur Realisierung zur Verfügung.

Quality: Product modifications after delivery will result in high service costs and are to be avoided.

Experience: Experience in using ANSI-C on different hardware platforms is available. No experience with ARM architectures, the use of an RTOS system or architectural design tools.

Budget: There is a small budget for the purchase of development tools.

Selection of a development environment based on the abovementioned requirements:

As the project will never require more than 256 KROM based on a single chip design, the low-cost Keil RV ARM compiler variant, with memory limitations, is selected as a basis for the development environment so as to achieve higher flexibility in spite of the small budget.

The analysis of the architectural design shows only few tasks with different concurrencies. Complex extensions are not very likely in spite of the scheduled lifetime. This implies a simple runtime architecture which can be easily implemented through a main() loop. For this reason, and in view of the training period, no RTOS will be used. Real-time tasks with defined reaction times are implemented at interrupt level.

In this environment, software will be developed with conventional methods, i.e. manual C code generation. Nevertheless, the ANSI C code is required to be technically correct. Code quality requirements are met by integrating a static code analyzer (PC-Lint) in the development environment.

In view of the required robustness at the time of delivery and only few further developments, it was decided to do manual testing with the Keil µVision debugging environment and U-Link debugging interface.

Longevity requirements and possible minor adaptations demand good documentation that enables software engineers to understand the software and use it safely and efficiently. To meet this requirement, minimize the documentation effort and rule out inconsistency between code and documentation, we decided to use the documentation tool Doxygen.

Due to time pressure and the lack of experience with ARM architectures, a 3 days ARM training is scheduled so as to minimize the time to get familiar with the hardware architecture.

All things considered, the chosen environment is characterized by minimum training effort. No unknown technology is used except for the ARM hardware, thus, unwanted delays during implementation are not to be expected.

Even though the budget for the development environment was small, we could still include an ARM seminar and the purchase of PC-Lint, based on the low-cost, limited compiler version (which does not imply any limitations in this case).

Components:

- Keil RV ARM compiler with memory limitations
- PC-Lint
- Doxygen (1)
- U-Link
- Eva board
- ARM training (3 days basic)

Total cost 4.000 EUR

(1) Doxygen is not included. Use as GNU General Public License.

We recommend the use of a configuration management tool with the abovementioned environment.

Variant 2

This variant is based on the following customer-specific requirements:

Hardware architecture: The quantity production of the application is approx. 10.000 pieces per year at a product price of approx. 400,- €. A single chip architecture is planned for budget and EMC reasons (no external memory). The system features complex operation and multilinguality resulting in increased memory requirements. Therefore, it is not yet clear if the concept will be extended with external memory.

Real-time requirements: High only for two concurrencies. However, there are further concurrencies, such as a complex GUI and several communication channels that shall be implemented as a bus system (including CAN and USB).

Product lifetime: 7 years. According to experience, requirements will increase over this period and there will be further developments. Maintainability and extendability shall be guaranteed over the whole lifecycle.

Time-to-market: Market launch will take place in 11 months. Two software developers are available.

Quality: System failure is not dramatic and can be corrected. Failure shall be minimized nevertheless to safeguard the company's reputation.

Experience: Experience in using ANSI-C on different hardware platforms is available. No experience with ARM architectures, the use of an RTOS system or architectural design tools. Microsoft Visual SourceSafe is used as configuration management tool.

Budget: Recent experience has shown that complexity and quality requirements could not be met with previous methods. Budget is available for the purchase of standard development tools (compilers, debuggers, eva board). In addition, management can be convinced to invest in required software engineering measures if the related costs are adequate and if they can be implemented without affecting development times.

Selection of a development environment based on the abovementioned requirements:

The environment is designed for the development of applications with moderate complexity. System reusability and maintainability are enhanced by using an RTOS supporting the runtime architecture design. The RTOS moreover simplifies interfacing to a decoupled hardware abstraction layer through asynchronous communication mechanisms.

Keil/ARM RealView Real-Time Library was selected for this variant. It contains both an RTOS and CAN and USB device interfaces.

This environment strikes a balance between mere C-level programming and UML level modeling. The RTOS supplies useful mechanisms for the architectural design while software is still programmed in C.

It only takes a few days to get trained on an RTOS system. In many cases, the time spent for training can even be recovered by creating an architectural design during the tailor-made RTOS start-up workshop. In view of the short development time, a training on CAN and USB drivers was also scheduled. In addition, it was considered to outsource the realization of CAN and USB to external service providers should there be a risk of the product launch date being delayed.

The Embedded Display Builder was used for an efficient design.

Components:

- Keil compiler IDE
- Keil Real-Time Lib
- Evaluation board
- ARM training (full)
- RTOS start-up workshop
- ARM real-time lib workshop
- Embedded Display Builder

Total cost approx. 16.500 EUR

A comparable solution is also possible based on IAR ARM combined with embOS as runtime environment.

Variant 3

This variant is based on the following customer-specific requirements:

Hardware architecture: The planned project will be produced with medium scale quantity only (10.000 per year). The final price is approx. 2.500,- € per device, therefore, the cost for the processor architecture is also an issue. Nevertheless, product management decided not to save on memory and processing performance.

Runtime architecture requirements: There are concurrencies, some of them with real-time requirements. The product will be available in several variants and with several operating modes (service and diagnostic mode, calibration mode, standby mode, data synchronization mode, loading mode, ...) The resulting dependencies and special states usually imply high complexity.

Product lifetime: approx. 8 years. According to experience, requirements will increase continuously over this period. Further developments shall be based on delivered products and have to be upgradeable. Maintainability and extendability shall be guaranteed over the whole lifecycle.

Standards: A semi-finished part is manufactured which several end customers will integrate in their products. Some customers require compliance with the MISRA standard. Interfaces, such as CANopen, have to be defined and observed.

Time-to-market: Market launch is scheduled in 12 months. 4 software developers are available. So far, there have always been last minute changes to accommodate the operational concept or other issues. The customer has not been able to reduce the amount of change requests in spite of accurately created specifications.

Quality: System failures might result in major recourse claims. There have been recurring quality problems in the past which shall be eliminated in future.

Experience: Experience in using ANSI-C on different hardware platforms is available, however, no experience with ARM architectures. There is some experience in using an RTOS or architectural design tools.

Budget: Budget is available for the purchase of development tools. In addition, management can be convinced to invest in required software engineering measures if they help safeguard quality and delivery reliability.

Selection of a development environment based on the abovementioned requirements:

This demanding environment for complex development projects sets a high value on software quality and reusability which shall be guaranteed over many years.

Particular attention is paid to standards like MISRA, to UML and a HAL (hardware abstraction layer). At the same time, this environment already considers upcoming standards (e.g. AUTOSAR, SPICE, IEC61508).

It meets all requirements for the so-called model driven development approach. Basic functions can thus be developed quickly as prototype.

UML

Due to the many operating modes and concurrencies, software engineering focuses on the architectural level. The architectural design is modeled with the UML notation. UML is increasingly used as standardized modeling language in the embedded industry as well.

The use of UML has many advantages compared to conventional software development:

- Long-term understandability and maintainability of the generated software
- Improved project documentation that is understandable also for parties from outside the project
- Consistency of model, code and documentation
- Easy reuse of components in different variants based on object oriented encapsulation
- Clear definition of interfaces to “deeper“ parts of an application, e.g. the hardware abstraction layer

Code Quality

As several customers have demanded compliance with the MISRA standard, a runtime environment and framework for Embedded UML Studio was selected which complies with MISRA 2004.

The quality and MISRA compliance of new code is verified using the static analyzer PC-Lint.

Based on ULM, many tests are automated much more easily because there are test tools for the interpretation and control of UML models.

Time-to-Market

The implementation of this project involves extreme time pressure, and a high degree of complexity is expected. In the past, the project has been slowed down over and over due to last minute change requests. For this reason, the environment relies on rapid prototyping. The scope of delivery of this environment comprises an evaluation board from Luminary Micro including a set of driver libraries. Correspondingly, a hardware abstraction layer HAL was specified at UML level and implemented exemplarily for some components of the hardware periphery.

Thus, a prototype can be generated quickly in order to test the interfaces and operational concept. A decision on the final hardware configuration can be made later in the project, and only the lowest driver level has to be adapted then. It will be just as easy to use components from one application in other projects. They are optimally decoupled from the hardware and runtime architecture through HAL. This simplifies product variant management, and further product variants can be realized within a short time.

Training

In spite of the complexity of the development environment, training will not take longer than 2-3 weeks. Experience has shown that, due to the enormous increase in efficiency across all phases of the development process, both financial and time-related investments are going to amortize within a few months. To safeguard the short implementation time, a 5 days on-site coaching is scheduled.

Compilers and IDE from IAR were already used in previous projects, so that experience is available, and the development environments remain homogenous across different projects.

Components:

- Embedded UML Studio
- Bridge UML to IDE (Bethooven)
- IAR compiler IDE
- embOS RTOS (object code)
- ATMEL eva board
- PC-Lint
- ARM training
- Embedded UML start-up training
- 5 days project start-up coaching
- ARM training

Total cost approx. 28.000 EUR

Variant 4

This variant is based on the following customer-specific requirements:

Hardware architecture: The planned project will be produced with medium scale quantity (approx. 1.000 per year). The final price is approx. 5.000,- €. Thus, the cost for the processor architecture is an issue but can be neglected compared to items like manpower costs. Sufficient memory and processing performance will be made available.

Real-time requirements: There are concurrencies with high real-time requirements. Most of the application, however, has insignificant real-time requirements. The product will be available in several variants.

Intellectual Property (IP): The product will incorporate a high amount of IP. Past experience has shown that market leadership was achieved through direct cooperation with the customers and a careful consideration of their requirements as to the design of functions. Most of the information and know-how on solution approaches and functions is in the heads of the developers, which makes it difficult to integrate new team members in the development process. Information shall be documented more efficiently in future.

The abovementioned procedure moreover leads to permanent changes. There is no completed design phase with a subsequent implementation phase. The problem is not that the current product requires new features but that the development department can no longer guarantee permanent further development based on the existing software.

Product lifetime: 10 years (the current product has been in the market for 20 years). According to experience, requirements will increase continuously over this period. Further developments shall be based on delivered products. Maintainability and extendability shall be guaranteed over the whole lifecycle.

Time-to-market: There is no concrete schedule. Considerations rather focus on issues like: How can a smooth confluence of new development and maintenance of the current system be achieved? To what extent does it make sense to reuse existing sources? ...

Quality: System failures might result in a loss of image. For now, the company is known for the robustness of their systems, especially regarding operational concepts.

Experience: Experience with the use of ANSI-C on different hardware platforms is available, however, there is no experience with ARM architectures, RTOS or architectural design tools. Telelogic Synergy® is used for configuration tasks, and Telelogic Doors® will be used as requirements engineering tool.

Budget: Management is aware of the fact that software engineering is of vital importance to the company. Major investments will be made if they help to safeguard IP and quality.

Selection of a development environment based on the abovementioned requirements:

This environment is state of the art and comprises all components of variant 3. In addition, this variant facilitates the automation of tests at model level, so-called regression tests.

In a further step, the architecture can be simulated based on a web interface at an early project stage. After a short time, an executable model can be generated through the so-called MDD approach (model driven development). First acceptance tests, e.g. with users or product marketing, can already be run on this model.

Quality assurance:

More and more developments are required to offer quality assurance according to certain standards, especially for safety-critical applications. This environment provides the required means for quality assurance according to EU standard IEC 61508, from requirements engineering to test automation and requirements traceability across the whole project. Compliance with the MISRA 2004 programming standard moreover accommodates the technical quality of the actual code.

Intellectual Property (IP): Requirements engineering is used to safeguard Intellectual Property. This variant offers all means for working with a requirements engineering tool. At the same time, tests are automated based on requirements. This environment facilitates state-of-the-art requirements, software and quality engineering.

However, the related integration will not only change the tools and notations etc. Engineers will also have to change their approach to work and methods as well as their way of thinking and have to strictly adhere to process phases as defined in the V-model. This not only involves getting trained in new techniques but also getting familiar with new approaches by means of different training measures that are suitably scheduled over several months for the different project phases.

Of course, the changes as described before will require additional time. However, this effort is justified in view of IP protection, quality standard and short innovation cycles on a long-term basis.

Components

- Embedded UML Studio™
- Bridge UML to IDE (Bethooven)
- TestConductor
- ValuePack
- Embedded OO-RTX™
- Keil compiler IDE
- PC-Lint
- Embedded UML start-up training
- 2 days requirements based project start-up workshop
- 2 days UML project start-up coaching
- 2 days TestConductor start-up workshop
- ARM training

Total cost approx. 42.000 EUR

General Information

We basically recommend you to use the following tools which are also available as open source products:

Configuration Management

The use of configuration management tools is recommended, offering substantial benefit and hardly any disadvantages.

Many tools are available to this end. Unless specific requirements have to be considered, the use of sub-versions as open source variant, for example, is a good way of not overstretching the budget for development tools.

Programming at C Code Level

Consistency between specification (requirements), documentation, code and tests turns out to be one of the major issues of modern software engineering approaches. For this reason, we basically recommend tools that provide for consistency of all required software engineering documents. Using the UML notation with a good UML tool is the best approach to this end.

When programming at ANSI-C level, development teams can keep documentation and code consistent with the following tools:

- Graphic editors enhance the understandability of code, e.g. EasyCode. Documentation can be created directly in the source code. It is basically hidden but can also be displayed if required. Code itself is not inflated with excessive text and remains understandable. Parallel documentation is not required any more. Documentation is not available in a separate document which is a disadvantage of this approach.
- Certain tools support consistency of documentation and code by automatically creating documentation from the source, for example Doxygen.

WILLERT SOFTWARE TOOLS GMBH ■

Tel. +49 5722 9678 60 - FAX: +49 5722 9678 80
Email: info@willert.de - www.willert.de
Hannoversche Str. 21 - 31675 Bückebug