



November 2008

This newsletter is brought to you courtesy of Art of Technology, a leading European specialist for customer specific electronic system design and development in hardware, software and electronic miniaturization.

Art of Technology will be exhibiting at **electronica 2008** in Munich. Please visit us from November 11th to 14th at our **Booth 157** in **Hall B4**.



Visit also our homepage www.aotag.ch

Content

Art of Technology

Technologies

Everything you always wanted to know about dependable and safety-critical systems programming

Application

See our new flyers of Art of Technology customer applications

Upcoming Events

Art of Technology AG

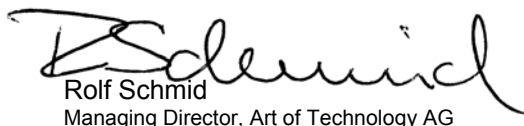
Art of Technology offers the whole spectrum of services for electronics design and development in hardware and software. We work together with our customers and support their team with exactly the processes they need, or take over a whole system as turn-key project.

Especially Art of Technology's expert know-how in medical technologies and sensors combined with High Density Packaging (HDP) technologies for a cost optimized system miniaturization of electronic systems is nearly unique. This allows us to realize innovative solutions together with our customers.

Art of Technology is certified according to ISO9001 and ISO13485 (even for active medical implants).

Please visit our new homepage on www.aotag.ch and learn about the services and support Art of Technology can offer you.

Yours sincerely



Rolf Schmid
Managing Director, Art of Technology AG

Technologies

Everything you always wanted to know about dependable and safety-critical systems programming

from Dr. Thomas Gillen and Rolf Schmid, Art of Technology AG

Software design for dependable and safety-critical is error prone and its outcome sometimes even dangerous. If only a few basic rules are followed, life can be made much easier and more reliable.

An ever increasing variety of embedded systems are available and in use, and their number grows dramatically. Even dependable and safety-critical systems are nowadays implemented by use of small micro controllers. These systems are different than standard embedded systems: operational failure of such an embedded system affects the real world. Serious damage, injury and even lost of lives can happen. Therefore such systems are developed by the use of rigid development processes to ensure careful design, implementation and test.

There are two major topics, which affect the programming of safety-critical embedded systems: the system requirements and the implementation of such systems.

An important impact to the dependability of safety-critical systems is the definition of the system requirements. Software driven systems are often very flexible and configurable. Additional features to the main functionality increase interoperability, adaptability and even comfort. Market(ing) needs require such functionality also for safety-critical systems. But, every function and every configurable parameter increases the complexity of the system. Every additional feature not required, but even considered as nice to have, must be well defined, specified, implemented, tested and even maintained. Even if the feature itself is not a safety-critical one, unexpected side-effects could introduce negative impact to the main, the safety critical functionality: Comfort and flexibility lead to a large-scale decrease of safety.

This brings up a rule that is not very popular:

"Reduce complexity and functionality of safety critical systems to the bare bones.
Small is beautiful, especially for safety critical or dependable systems."

If additional functionality is really required, the old rule of "Divide and Conquer" has to be considered. Limit the functionality of the safety critical part and encapsulate it in an isolated part of the device. Then put all other nice to have features in a separated part of the system and link both by a small but well defined interface. The additional effort pays off by reduced complexity, reduced effort spent for specification, design, test definition, code inspection, testing and even certification of the safety critical functionality.

Another often-underestimated part of the development is the implementation, because there is a subtle issue often not considered properly and which could lead to subtle errors.

The use of a high level language for the programming of embedded systems is widely accepted, and use of assembler language is limited to some special cases. Only a few programming languages are in use, C and C++ cover almost 98%. Due to the cost constraints of most embedded systems, the smallest and most inexpensive processor that will do the job is normally used. This means limited resources for memory and computational power. This limits the use of C++ due to its demand in resources even to larger embedded systems. Almost all smaller embedded systems are therefore implemented in C. However, for several reasons, C is not the best choice for programming of embedded systems (but there is no better one available and accepted). The C programming language

The C programming language was originally developed in the early 1970s for the implementation of the UNIX operational system, and its main focus was on compact and fast code. This makes it also attractive for embedded system programming. But this is also a disadvantage: C-compilers usually don't add any code for runtime checking or integrity enforcement. Therefore, all such checking is in the own and sole responsibility of the designer and the programmer.

Another critical point is, that the language definition contains some ambiguities. Usually, this is solved compiler specific, but this leads to unexpected and implementation dependent behavior of the produced code – a variety of dialects saying "mainly" the same.

In addition, C is a very powerful language with only little restrictions that give programmers an incredible freedom of code writing. Most programmers are not aware, how sharp this knife called C is in reality, because only a few programmers understand C and its implementation in full depth. Due to its flexibility, unusual use of the language is possible. Such code conforms well to the written standard, but it is hard to understand and to inspect. So it is very easy to write "dangerous" code, and such hidden errors could surpass even code reviews and code inspections. The freedom of the language is a big weakness with regard to clean and safe

code.

Finally, the wide use of C for many different processors decreases the error coverage. There is a large number of compilers in use, and the number of experienced users per specific compiler is therefore quite small. As a consequence, compiler errors often survive for quite a long time. Even known errors remain unfixed, even for several compiler generations.

To overcome this limitations, coding guidelines and rule sets are established, to ensure at least a "reviewable" code. Almost, such guidelines are created by a single person or a small team with focus to reuse or review, but usually not with deeper knowledge of the hidden pitfalls of C.

MISRA, the Motor Industries Software Reliability Association has therefore issued a set of more than 120 rules and recommendations, generated by very experienced programmers, to reduce the risk of using C. This standard is well documented and known as MISRA C. It is a subset of C and tries to avoid the pitfalls induced by language ambiguities and other implementation specific solutions trough a very restricted usage of the language. The standard and some accompanying documents describe also in detail, why specific rules are mandatory, or the background for the recommendations. Even experienced programmers with years of practice are surprised about some explanations. Many unexpected behavior seen in the past, which could never explained in detail, is now understandable.

The big advantage of MISRA C is its independence of the development system. No special tool is required to write MISRA C conforming code, so almost all compilers and development systems are usable. There are some tools, which check existing code for conformance with the rule set, and the newer IDEs of some suppliers already integrate such tools for online checking of the code.

Even if MISRA C is not the ultimate solution, and even if compilers remain buggy, the usage of the MISRA C rule-set is always a good starting point for programming of dependable embedded systems.

System requirements definition, design, programming and test of software for critical embedded systems remain a challenging task. Reduction of requirements helps to lower the complexity and the effort spent to the accompanying tasks like specification, implementation and test. Tools like MISRA C could help to make implementation and test a little bit easier, but there is no easy way to guarantee correct code. It is always in the responsibility of the company to create an environment that encourages employees to reduce the complexity and to produce clean and safe code.

Application

Please find our new flyers of customer applications under
http://www.art-of-technology.ch/english/publications_flyers.html

- POKEN
- PermaSense

For further information visit us at electronica 2008 in Munich, November 11th – 14th, 2008, Booth 157 in hall B4 or contact us at: info@aotag.ch

Upcoming Events

electronica 2008
(AoT will be exhibiting)
November 11th–14th 2008
Munich, Germany
www.electronica.de/

hybridica 2008
November 11th–14th 2008
Munich, Germany
www.hybridica.de/

Compamed 2008

November 19th - 21st 2008
Messe Düsseldorf, Germany
www.compamed.de/

SPS/IPC/DRIVES 2008

November 25th – 27th 2008
Nurnberg, Germany
www.mesago.de/spsp

Embedded World 2009

March 3rd – 5th 2009
Nurnberg, Germany
www.embedded-world.de

Medtec 2008

(AoT will be exhibiting)
March 3rd - 5th 2009
Landesmesse Stuttgart, Germany
www.medtecstuttgart.de/

Hannover Messe 2009

April 20th – 24th 2009
Hannover, Germany
www.hannovermesse.de

Unsubscribe: If you don't want to get this newsletter in the future,
please send us a short e-mail to newsletter@art-of-technology.com