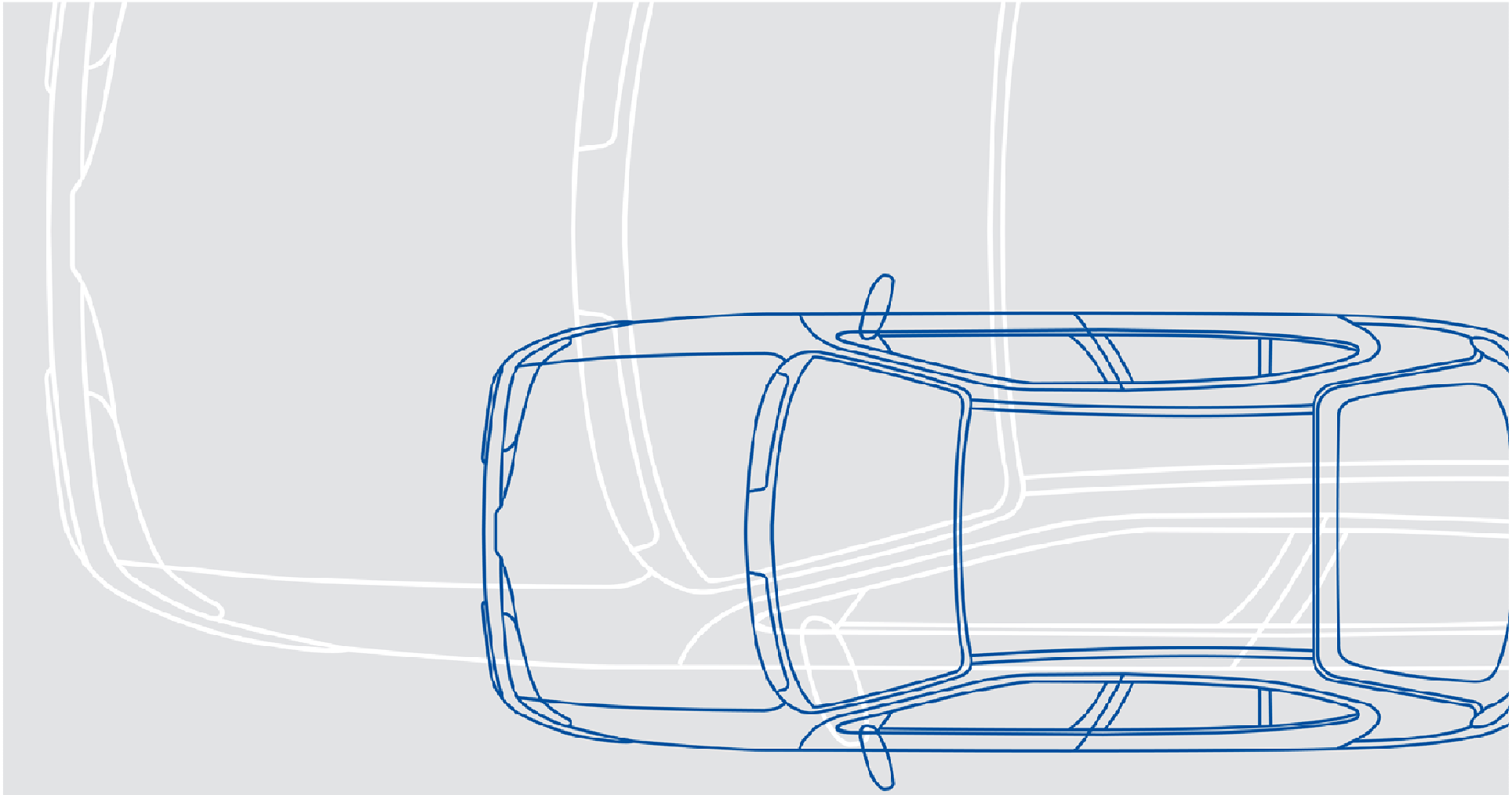
Design of safety critical systems with ASCET



Safety critical systems with ASCET: Agenda

- Limitations of C
- Model based design – Overview and Advantages
- Introduction to ASCET
- ASCET models vs. UML
- ASCET models vs. Ada
  - Fixed point arithmetic
  - Exceptions
- Challenges associated with Model Based Design approach
  - Validating the code generator
- Summary
- References

Safety critical systems with ASCET
Limitations of C language

- Behavior depending on the bit size of the int type: u32A = u16A + u16B will produce different results for 16 and 32 targets
- Weakly defined order of evaluation: in u32A = getX() + getY() either function call may be executed first
- Implicit loss of sign due to integer conversion: u32 < s32 will convert s32 to unsigned long
- Counter-intuitive precedence rules: a + b << 1 first computes the sum that is then shifted by 1, instead of treating the shift like a multiplication.

Safety critical systems with ASCET
Model Based Design - Overview

- The development starts from making mathematical models. These models can be used for...
  - Validation of the algorithms through PC simulation
  - Simulation in real time target
  - Generate automatic code

- Why this approach ?
  - Gaps due to technical representation and understanding..
    - There exists a huge gap between the function developer and software engineer who implements the algorithm. Paper specifications are not used. Models are developed by control function developers and given as an executable specification.
  - Need for reducing the development time.
    - Time to market is a very important in today's scenario and this is very well achieved by autocoding.
  - Avoid human errors
    - Human errors are avoided in coding due to autocode. Unlike human errors, the errors injected by code generator are systematic and not random. Systematic errors can be removed systematically.

Safety critical systems with ASCET
Further advantages of Model Based Design

- Early validation of concepts through simulation and rapid prototyping gives enough room for development of many new algorithms. This is very useful for function developers (This is explained in detail in the next slide)

- Standard compliance
  - Code generators are generally designed to meet the standards which are specific to an industry. e.g. MISRA-C:2004 compliant autocode is generated for automotive use case. (Motor Industry Standard Reliability Association – MISRA has given set of rules for using C language for automotive application) http://www.misra.org.uk/

- Traceability is very important for the software with respect to the functional requirements.
  - Traceability with model is easier compared to the code

- Documentation
  - Model based design tools come with automatic documentation generation facilities. The documentation is most of the time self explanatory. Ambiguities are resolved due to executable specification.

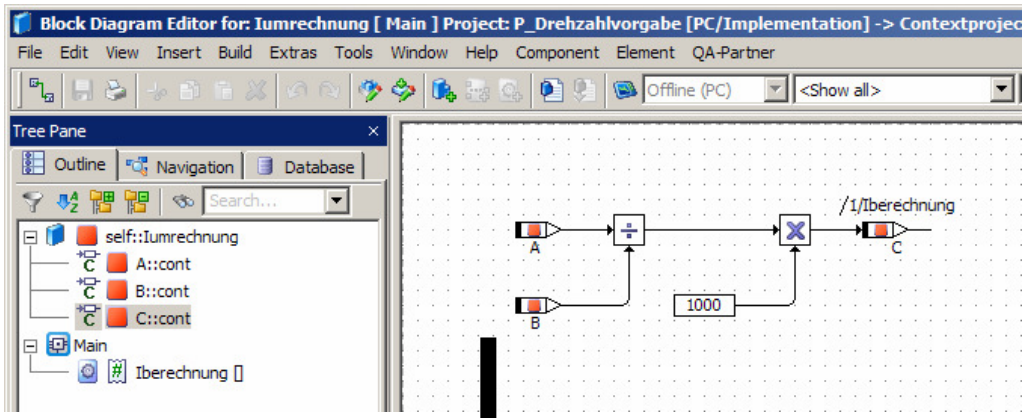
Safety critical systems with ASCET
Early validation techniques supported by Model Based Design

- Function models can be validated during early phases of development through simulation. Following are the methods which are generally used for validating the function models.
  - Model in the Loop simulation (MiL)
    - Validation of control functions through simulation of floating point models
  - Software in the Loop simulation (SiL)
    - Validation of production quality C code (ECU specific code) through simulation
  - Rapid prototyping
    - Validating the models in a real time execution target.

- Need for early validation
  - Errors in software algorithm and implementation errors can be easily identified during the early phases of development. Hence probability of identifying the bugs at a later point in time is significantly reduced resulting in a better quality.
  - The cost for fixing a detected problem increases by the factor 10 for each process step it doesn't get detected.
  - 60% of automotive electronic systems development cost is contributed by the software development.

Safety critical systems with ASCET
Introduction to ASCET

Specify the model: domain specific language for the automotive industry

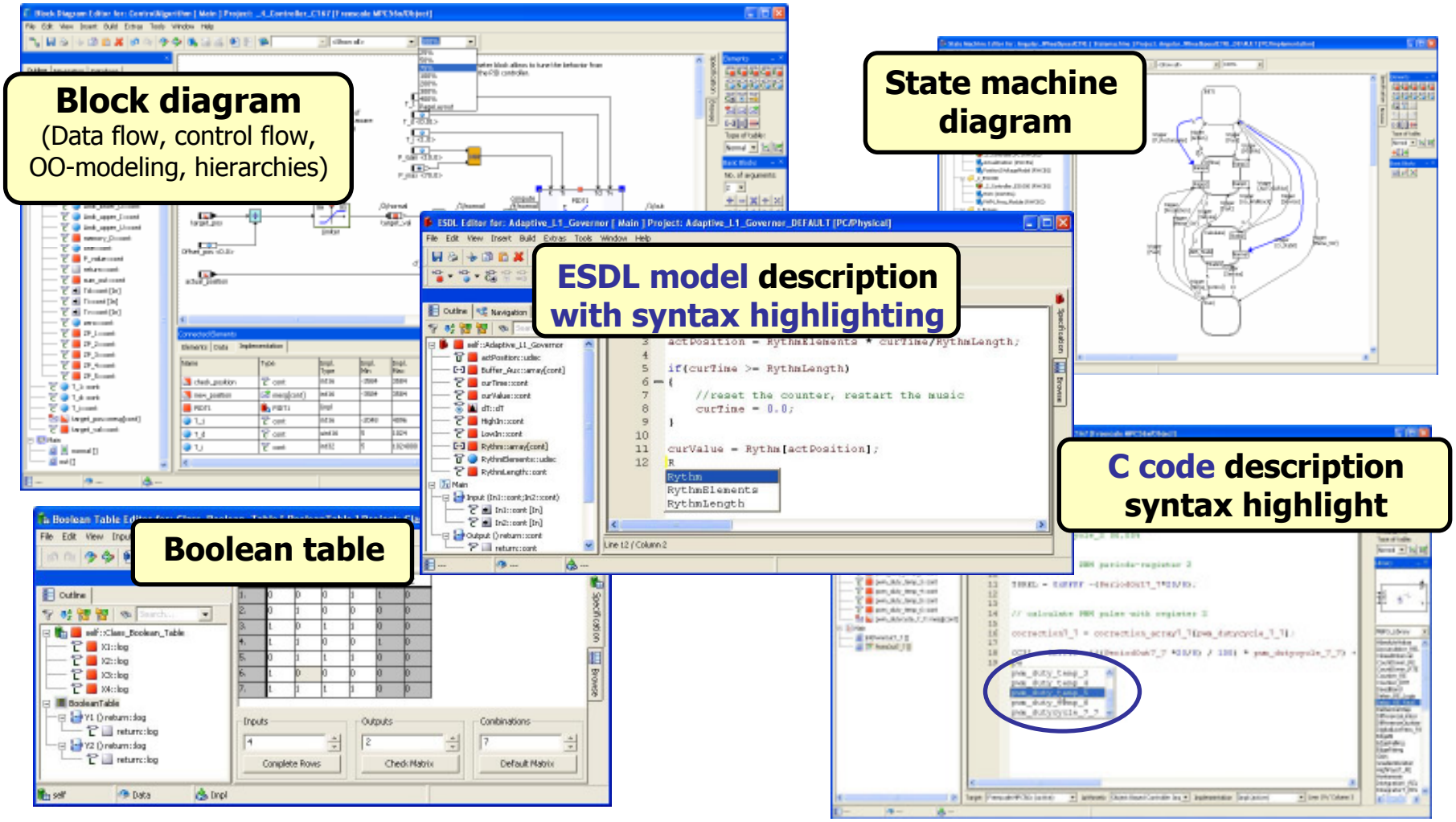


Generate code

```
/* public Iberechnung [] */
void IUMRECHNUNG_IMPL_Iberechnung(void)
{
  sint32 _t1sint32;
  /* Iberechnung: sequence call #1 */
  _t1sint32 = ((uint32)(IUMRECHNUNG_IMPLinstance->A->val * 125) >> 7) /
              IUMRECHNUNG_IMPLinstance->B->val;
  /* assignment to C: min=0, max=65535, hex=4phys+0, limit=(maxBitLength:
true, assign: true), zero incl.=true */
  IUMRECHNUNG_IMPLinstance->C->val =
              (_t1sint32 <= 15) ? ((uint32)_t1sint32 << 12) : 65535;
}
```

Run code in simulation on a PC or deploy on embedded controller

Safety critical systems with ASCET
Different types of Models



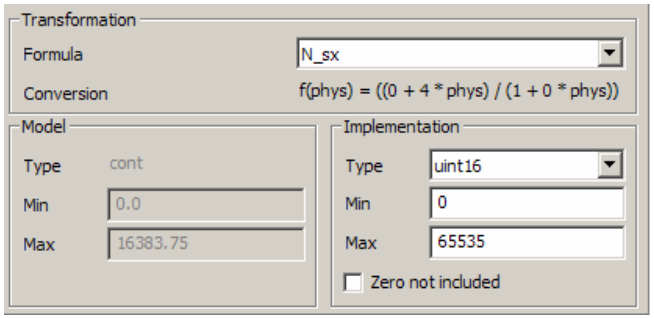
Safety critical systems with ASCET
Fixed point arithmetic

Embedded control units are resource constrained:

Floating point arithmetic is expensive => Fixed point arithmetic is used

The model contains the specification of the variables:

Value range, Precision and Data type



Calculation is specified on the physical model:

c = a + b;

The code generator take care of the implementation details

Safety critical systems with ASCET
ASCET vs. UML

UML is primarily a design notation:

- Many different diagrams on various levels of abstraction
- Language independent
- Does not contain executable behaviour

ASCET models:

- Fewer diagrammatic styles, and no higher-level abstractions like package or deployment diagrams
- Have both intrinsic value and added value in combination with code generation
- Are executable on multiple different platforms
  - From a 64-bit PC to a 16-bit microcontroller
- Natively supports the C programming language

Safety critical systems with ASCET
ASCET vs. Ada: Fixed point types

Ada has a strong type system

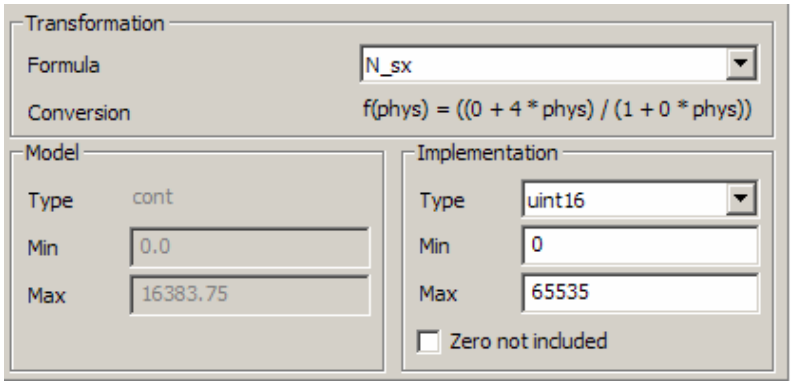
This extends to fixed-point, requiring different types for values of different precision.

Precision/intervals are specified, data type is chosen by the compiler.

```
type VOLT is delta 0.125 range 0.0 .. 255.0;
```

ASCET contains one generic model type "continuous"

- Equivalent to "real" in Ada
- Precision/intervals and data types are specified.
- Model type is realized as "fixed"
- or "float" in the implementation



Safety critical systems with ASCET
ASCET vs. Ada: Fixed point arithmetic

Ada:

- Arithmetic between fixed point types is only allowed if they have the same precision, except for multiplication and division, where the target precision must be specified.
- Semantics are specified by the LRM, but complex (compiler is only required to provide *at least* the specified precision - the "small" of the type)

ASCET:

- Arithmetic between all "continuous" expressions is allowed. The code generator takes care of the details: overflow protection, re-scaling, selection of precision in complex expressions.
- Semantics are defined by the code generator
  - And the code generator is proven by use

Safety critical systems with ASCET
ASCET vs. Ada: Exceptions

Ada throws runtime exceptions in dangerous situations:

- Array index violations
- Division by Zero
- Integer overflow
- Assignment interval mismatch

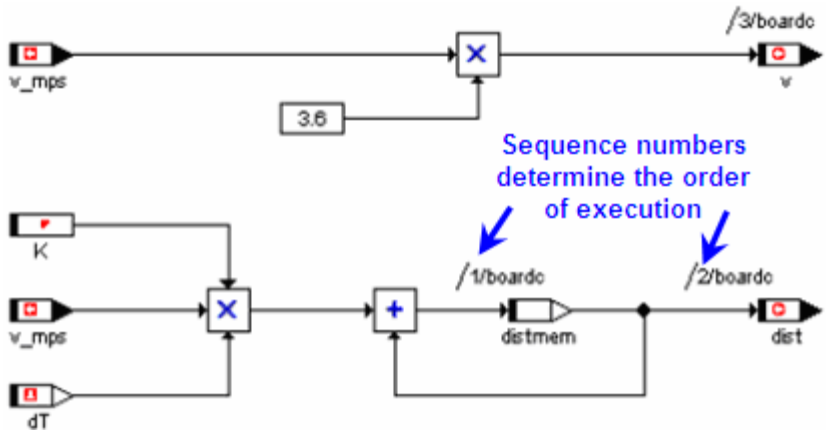
The ASCET code generator implements domain-specific default behavior:

- Division by Zero is protected, returning the max value
- Integer overflow is avoided
- Saturated arithmetic on specific microcontrollers is supported
- Assignments are limited to the specified range where necessary
  - Array indices are not limited – cannot assume a default behavior.
  - Array index violations are not expected to occur in practice due to checks in the model or extensive testing.

$$\left\lceil \frac{a}{2} + \frac{b}{2} \right\rceil_{0}^{2^{32-1}} \times 2$$

Safety critical systems with ASCET
Simulation behaviour versus ECU code behaviour

- Sequence numbers in ASCET give complete control over the model and hence the code.
- This ensures the ECU code behaviour to be same as the simulation behaviour.
- For ECU code generation, the message duplication is handled automatically by ASCET.
- Human errors are avoided.



Safety critical systems with ASCET
Challenges associated with Model Based Design approach

- Simulation and implementation:
  - Function development may use all possible components/blocks including the continuous time blocks which are not suitable for code generation. Final stage of code generation requires high level of model customisation to fit the memory and run time requirements of the ECU.

- Code optimisation versus model optimisation:
  - optimising the model for achieving a particular RAM/ROM/Runtime is an important phase in the development and there exists no universal solution for achieving this.
  - The code generator would have some flexibility to achieve this, nevertheless in practice the engineer has to decide carefully on a case to case basis.
  - Industry practice is to have a combined approach of reviewing the model and the code together and optimising the model. This would take at-least a few iterations and the behaviour of code generator plays an important role here.

- Domain specific compliance
  - Example: Compliance with MISRA-C:2004 standard for automotive applications.

Safety critical systems with ASCET
Challenges associated with Model Based Design approach

- A given logic can be implemented in many ways in the model and this is similar to the C coding patterns/style which is specific to a particular engineer. Different models will give rise to different C code.

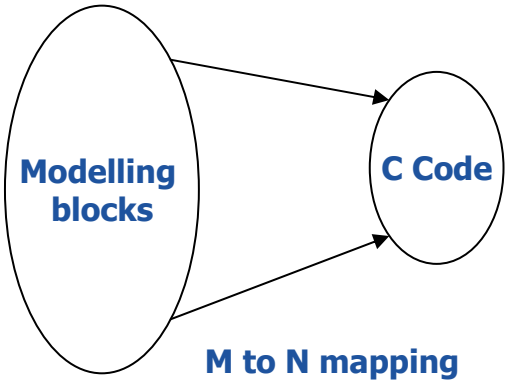
- The code generator has to generate a code for various permutations and combinations of the modelling blocks. Since the mapping algorithms has to handle many blocks and generate C code using finite C constructs, the mapping cannot be efficient unless the number of blocks are less.

  - Standard industry practice is to use modelling guidelines
  - Also model libraries are developed and maintained by development teams to standardise the process.

  - Similar to the C coding, model development also demands specific skills

  - Function development team would focus mainly on the algorithm

  - Code generation team would customise the models for ECU code generation



- One advantage is that the for a given model construct the code generator will always consistently repeat the code style.

Safety critical systems with ASCET
Validating the code generator

- Manually reviewing the code:
  - Compare the code with the model to validate the correctness of the code generator.
  - Automation is possible here. A given piece of code is compared with a corresponding model and the generated code structure is validated against the expected code structure.

- dynamic testing:
  - The generated code is executed with known inputs and the output is compared to an expected result. If a comprehensive set of test scenarios exists for a given model, they can as well be used to verify a new version of the code generator.

- The two above approaches have the inherent limitation to only verify the code generator for a specific model on a given target processor.

- In principle, it is not possible to transfer the results gained in a PC simulation to a rapid prototyping system or the final controller hardware.

Safety critical systems with ASCET
Summary

ASCET

- is a domain specific language for control algorithms in the automotive industry
- Enables early validation of algorithms using floating point arithmetic
- Offers complete control to specify the sequence of execution and hence no difference between model behaviour and code behaviour.
- Provides convenient fixed point arithmetic
  - Code generator makes sensible decisions for the usual problems of overflows, re-scaling etc.
  - Generation of fixed point arithmetic is done consistently
  - Models need to be tested to see if the achieved precision is sufficient
  - Fixed point settings are separated from the model and hence conforming to the ASAM-MCD-2C standard

References

- ISO/IEC 9899:1999, *Programming languages - C*, International Organization for Standardization, 1999
- Les Hatton, *Safer C: Developing software in high integrity and safety-critical systems*, McGraw-Hill, 1995
- Stephen Johnson, *Lint, a C program checker*, Computer Science Technical Report 65, Bell Laboratories, December 1977
- *QA-C : Advanced static code analysis for C*, http://www.programmingresearch.com/QAC_MAIN.html
- *Guidelines for the use of the C language in critical systems*, ISBN 0-9524156-2-3, MIRA, October 2004
- http://www.eclipse.org/projects/project_summary.php?projectid=modeling.emf
- Guidelines for the application of MISRA-C:2004 in the context of automatic code generation, ISBN 978-1-906599-02-6, MIRA, November 2007
- Ingo Stürmer, Mirko Conrad, Heiko Dörr, and Peter Pepper, *Systematic Testing of Model-Based Code Generators*, IEEE Transactions on Software Engineering, Vol. 33, No. 9, September 2007
- Diomidis Spinellis, *Global Analysis and Transformations in Preprocessed Languages*, IEEE Transactions on Software Engineering, Vol. 29, No. 11, November 2003
- http://www.asam.net