



Arm Development Studio

Revision: 2020.1

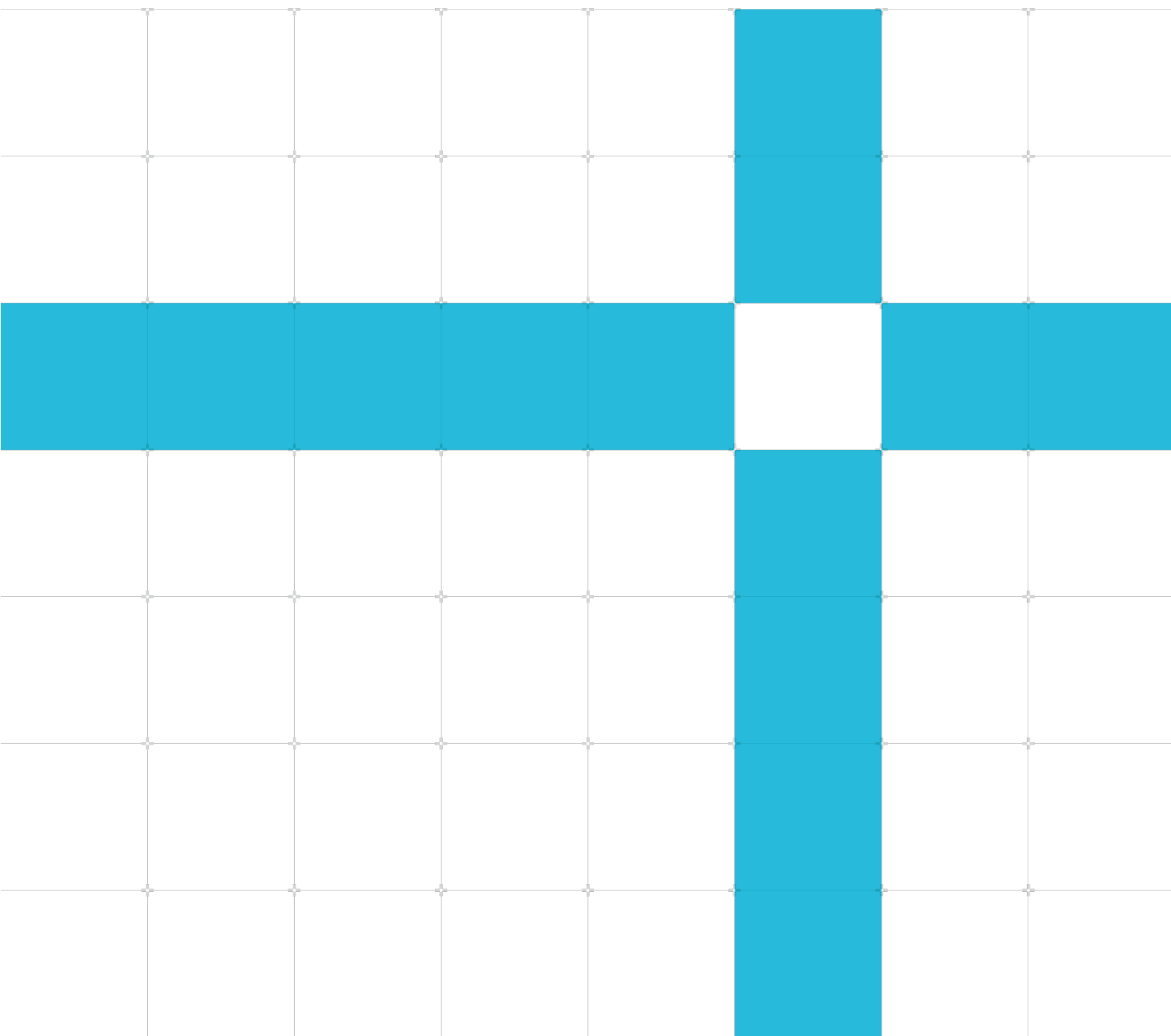
Optimizing ASCET-DEVELOPER generated code

Non-Confidential

Copyright © 2021 Arm Limited (or its affiliates).
All rights reserved.

Issue 01

Document ID: 102395



Arm Development Studio

Optimizing ASCET-DEVELOPER generated code

Copyright © 2021 Arm Limited (or its affiliates). All rights reserved.

Release information

Document history

Issue	Date	Confidentiality	Change
01	25 th January 2021	Non-Confidential	First release of document

Non-Confidential Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm. No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether implementations infringe any third party patents.

THIS DOCUMENT IS PROVIDED "AS IS". ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word "partner" in reference to Arm's customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

If any of the provisions contained in these terms conflict with any of the provisions of any click through or signed written agreement covering this document with Arm, then the click through or signed written agreement prevails over and supersedes the conflicting provisions of these terms. This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its affiliates) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow Arm's trademark usage guidelines at <http://www.arm.com/company/policies/trademarks>.

Copyright © 2021 Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20349)

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

Product Status

The information in this document is Final, that is for a developed product.

Web Address

developer.arm.com

Progressive terminology commitment

We believe that this document contains no offensive terms. If you find offensive terms in this document, please email terms@arm.com.

Contents

1 Overview 5

1.1 Before you begin..... 5

2 Code optimization 7

2.1 Use structures for messages and global data..... 7

2.2 Function arguments..... 8

2.3 Make use of intrinsic functions..... 9

2.4 Optimize placement of code and data..... 9

3 Related Information 11

4 Next steps..... 12

1 Overview

This guide describes how to integrate the ASCET-DEVELOPER development flow with Arm Development Studio.

A brand-new automobile might contain more than 100 million lines of code in the software that controls its various systems, from engine management to atmospheric control. This amount of code will only increase as automobiles become more complex. The increase in complexity of systems has led to widespread use of model-based control development tools, generating safety standard conforming, portable source code.

ETAS GmbH, a subsidiary of Robert Bosch GmbH, are a provider of development solutions for the automotive industry and related sectors. One of their product offerings is ASCET-DEVELOPER, a code generation tool which uses this type of model-based process. ASCET-DEVELOPER generates MISRA-C compatible application code, for use with the RTA-OS operating system.

Arm Development Studio is a comprehensive embedded C/C++ development solution for all Arm processors. Arm Development Studio includes the Arm Compiler, a mature toolchain tailored to bare-metal, firmware, and RTOS based applications. Arm Compiler for Functional Safety is a qualified C/C++ toolchain that has been assessed by the safety-accredited certification body TÜV SÜD. The qualified toolchain is suitable for developing embedded software for safety markets, including automotive, industrial, medical, railways and aviation.

The purpose of the integration is to provide a simple, iterative flow as control applications are developed and debugged. Optimization techniques are explored to tune the generated code to make use of the latest features of the Arm architecture.

This guide is intended for automotive or similar developers who are interested in a model-based software development flow. The tools described in this guide are Arm Development Studio and ETAS ASCET-DEVELOPER. However, content may be relevant to users of other tools, notably Keil MDK.

1.1 Before you begin

To work through this guide, you need to download and install the latest versions of Arm Development Studio and ASCET-DEVELOPER. The Arm and ASCET-DEVELOPER websites include basic installation and setup instructions. A valid software license is required for the tools to execute.

Arm Development Studio is supplied with the latest available version of the Arm Compiler at the time of release. If you want to use a different compiler version, for example Arm Compiler for Functional Safety, you can add this through the **Preferences** pane. For more information, see [Add new compiler toolchains to Arm Development Studio](#).

System initialization is highly device specific. CMSIS Packs provide initialization code, often with complete example projects, for many microcontrollers. If you are using a microcontroller that allows this, you can import the necessary packs from the CMSIS Pack Manager perspective of Arm

Development Studio. To import generic initialization examples for Arm processors:

1. Navigate to the **Development Studio Eclipse IDE**.
2. In the menu option, navigate to **File > Import**.

When you have created your Development Studio project, configure your ASCET-DEVELOPER project to generate code directly into the project folder. In the ASCET-DEVELOPER project properties:

1. Navigate to **Run > Run Configurations > Generate results into a folder**.
2. In the **Folder** field, specify the location of the project folder to generate to.

2 Code optimization

ASCET-DEVELOPER generates highly portable, MISRA-C compliant source code, which can be readily consumed by the Arm Compiler. We will introduce some useful tips to control how ASCET-DEVELOPER generates code to best suit the Arm Architecture.

Further information on global options for optimizing ASCET-DEVELOPER can be found in the User Guide. Similarly, the Arm Compiler User Guide contains general documentation on writing optimized code.

2.1 Use structures for messages and global data

The use of many global variables is generally considered bad programming practice. Data encapsulation and abstraction with structures and helper functions are encouraged to improve maintainability, readability, and correctness. Avoiding the use of many global variables is also beneficial for model generated code.

The Arm processor is a load-store architecture, and therefore the data processing instructions cannot access global data directly. This type of variable is accessed with a read-modify-write strategy, and use a register containing a base address, with an optional offset, to load and store data to the core registers.

Grouping global registers together in memory so that a common base pointer can be used is a common optimization for Arm. First, a base address is loaded, and then a read from that base address, with or without an address offset. We recommend a single base address and reference to many variables as an offset from that address. The preference for this practice leads to the preference for the use of structures for global data.

Because most embedded systems have multiple task rates, and pre-emption is used in many cases, any task container must take protected copies of messages, passed as variables between tasks. ASCET-DEVELOPER supports several types of protected message copy, for example to take copies of all used messages under suspended interrupts. You can choose task rates and define the execution of task containers within an OS configuration App file for any project.

The original code for the modules uses individual messages. This means that an individual update or read must be performed for each message for a task before any execution occurs. Because each task can manipulate many messages, this procedure can become quite time-consuming.

Because each module in ASCET-DEVELOPER has a group of output interface messages defined in an Embedded Software Description Language formatted data interface file, these can easily be replaced with a single structure of messages.

The following code shows an example output interface message:

```
data interface SPInterface
{
    @symbol("SP_Dig1Speed")
    T_Speed_SP_Dig1Speed = 0x0;
    @symbol("SP_Dig1Pin")
```

```
Boolean SP_Dig1Pin = false;

@symbol("SP_Dig1Duty")
T_DutyPercent_SP_Dig1Duty = 0x0;
...
```

The following code shows an example of replacement with a single structure:

```
data interface SPInterface
{
    SP_InterfaceData SP_If;
}
```

A memcpy can take a local copy for use in a task container. Because the size of the structure is well known, the compiler can select the most efficient memcpy implementation from the compiler supplied C library.

2.2 Function arguments

To aid creating scripts for ASCET-DEVELOPER class code unit testing, it is common to let each class have a single method which reads all inputs to the class. This method is called a ReadInputs method. This method has the benefit of generating a single line of source which clearly hooks into test scripts.

A disadvantage of the ReadInputs method is that the resulting code function has a non-optimal call. It is more efficient to pass a pointer to a structure rather than multiple arguments.

Each instance of each class has an internal static variable for each argument. It is possible to remove the ReadInputs method, and instruct ASCET-DEVELOPER to make copies of required data. To do this, select the **Set** option on the variables that are being read, as shown in the following screenshot:

Class Element

General	Static	<input type="checkbox"/>
Annotation	Name	SC_Gear
Comment	Type	types.T_TargetGear
	Kind	Variable
Visibility		<input checked="" type="checkbox"/> Set <input type="checkbox"/> Get
	<input checked="" type="radio"/> Private <input type="radio"/> Public	

Similarly, multiple output variables can be handled by selecting the **Get** option on the output variables.

The resulting C code replaces the ReadInputs function with structure-to-structure element copies which should be more efficient. This is because the structure base addresses need to be loaded only once.

2.3 Make use of intrinsic functions

The Arm C Language Extensions (ACLE) are a set of features to enable C/C++ code to exploit the advanced features of the Arm Architecture. One such extension is the definition of intrinsic functions that map to specific Arm instructions that may not otherwise be generated by the compiler.

ASCET-DEVELOPER allows the creation of an Arithmetic Services file that defines a function to be called in a specific scenario. That can subsequently be mapped to an intrinsic function at compile time.

For example, consider the following variable definition, which saturates at +/-32K:

```
representation R_Speed{
    range = -32768 .. 32767;
    datatype = sint16;
    formula = f_0bn0;
}

type T_Speed is real using R_Speed;
```

By default, this definition may result in inefficient source code being generated, with additional saturation code being applied, as seen in the code:

```
_t1sint32 = a+b;
x = (sint16)((_t1sint32 >= 32768) ? ((_t1sint32 <= 32767) ?
    _t1sint32 : 32767) : -32768));
```

The following code shows how you can define an arithmetic service so that a function is used to implement the code instead:

```
+1|s16|s16|s16=(sint16)ADDLIM_S16S16_S16(%i1%,%i2%)
```

The ASCET-DEVELOPER code generator will be able to make use of this function, for example:

```
x = ADDLIM_S16S16_S16(a, b);
```

You can then map this code to an ACLE intrinsic function in a user provided header file, for example:

```
#define ADDLIM_S16S16_S16(a, b) __QADD16(a, b)
```

This will compile into a single QADD instruction.

2.4 Optimize placement of code and data

The Arm linker uses a mechanism known as scatterloading to place code and data at appropriate addresses in your device memory map. You can use the ACLE `__attribute__((section(".name")))` feature to create special ELF sections which can be referenced by the scatter file. This can be used to place frequently used or high priority tasks in the fastest memory location. Many Arm devices support tightly coupled memory local to the processor for this purpose.

Within ASCET-DEVELOPER you can define MemorySections.xml to easily locate functions to particular regions. You may wish to define a section named TCMCODE for your functions to place in tightly coupled memory as you can see in the following code:

```
<MemClass>
    <name>TCMCODE</name>
```

```

<prePragma/>
<postPragma/>
<funcSignatureDef>%return_type% __attribute__ ((section (".ascettcm")))
    %gen_name% (%argv%)</funcSignatureDef>
<constQualifier>true</constQualifier>
<volatileQualifier>>false</volatileQualifier>
<description>memory section for code to execute in TCM </description>
<category>Code</category>
</MemClass>

```

Your scatter file can define an execution region for the tightly coupled memory and use the section name to easily place these functions within the code, as seen in the following code:

```

TCM_REGION    TCM_BASE    TCM_SIZE
{
    *(.ascettcm)
}

```

3 Related Information

Here are some resources related to material in this guide:

Product information

- [Arm Development Studio](#)
- [Arm Compiler](#)
- [ETAS ASCET-DEVELOPER](#)

Other resources

- [Arm C Language Extensions](#)
- [Add new compiler toolchains to Arm Development Studio](#)
- [Arm Compiler User Guide](#)
 - [Writing optimized code](#)

Industry bodies

- [TÜV SÜD](#)
- [Motor Industry Software Reliability Association \(MISRA\)](#)

4 Next steps

In this guide we have shown how developers can use ASCET-DEVELOPER model-based design tool to generate standards compliant source code for complex control algorithms. You can tune your design to enable the Arm Compiler from Arm Development Studio to build this code in an optimal manner for your Arm-based target. The Eclipse IDE provides a common environment, supporting third-party plugins for version control and other functionality. If you are new to either ASCET-DEVELOPER or Arm Development Studio, free evaluation licenses are available from ETAS and Arm respectively.

The process of developing code for automotive systems is at a turning point. Creating high safety integrity systems for ISO26262 or IEC61508 does not require a complex process, it requires a clean process with good traceability from end to end. Where possible as much of the process should be contained within one application.

Arm Development Studio, providing a FuSa qualified compiler, Eclipse based IDE allowing the use of the various version management and other plugins, and integrated debug support, coupled with model based graphical coding with ASCET, is on the path to such a process.