ETAS

RTA-OS V5.5
セーフティマニュアル

DRIVING | EMBEDDED EXCELLENCE

著作権

Document 10671-SM-5.5.3 R01 JP – 08.2017

目次

第1部 概要 7

1 概要 8

1.1 本書の対象ユーザー 8
1.2 本書の使い方 8
1.3 安全に関する注意事項 9
1.4 用語と略語 10
1.5 表記上の規則 12

2 ISO26262の概要 13

2.1 構成 13
2.2 ソフトウェア開発への適用性 13
2.3 ツールへの適用 14
2.4 ISO26262とRTA-OSとの関係 14
2.5 RTA-OS ISO26262認証の適用範囲 15

3 安全関連システム開発におけるRTA-OSの役割 17

3.1 RTA-OSの処理の概要 18
3.2 ISO26262とRTA-OSの関連性 20

第2部 RTA-OSツールの使用に関する考察 21

4 開発プロセスにおけるRTA-OS 22

4.1 他の補助プロセスとの相互作用 25
4.2 RTA-OSコンフィギュレーション 26

5 RTA-OSツールの使用に関する想定集 28

5.1 システムレベルの想定 28
5.2 RTA-OSツールの使用に関する想定 30

6 RTA-OSツールの安全性を証明する議論 35

6.1 RTA-OSツールの故障モード 35
6.2 RTA-OSツールの開発プロセス 37
6.3 安全性に関与するRTA-OSツール固有の機能 39

7 ISO26262に対するツールの評価 41

7.1 分析によるツールの評価 42
7.1.1 ツール信頼度（TCL） 42
7.2 分析によるツールの評価の要件 43
7.2.1 一意の識別番号（ISO26262-8/11.4.4.1a） 43
7.2.2 コンフィギュレーション情報へのアクセス（ISO26262-8/11.4.4.1b） 44
7.2.3 使用事例に関する記述（ISO26262-8/11.4.4.1c） 45
7.2.4 有効な動作環境に関する記述（ISO26262-8/11.4.4.1d） 45
7.2.5 最大ASIL（ISO26262-8/11.4.4.1e） 46
7.2.6 認定メソッドに関する記述（ISO26262-8/11.4.4.1f） 46
7.2.7 機能と技術的特質に関する記述（ISO26262-8/11.4.4.2a） 46
7.2.8 ユーザーマニュアル（ISO26262-8/11.4.4.2b） 47
7.2.9 有効な動作環境に関する記述（ISO26262-8/11.4.4.2c） 47
7.2.10 異常な動作条件下における挙動に関する記述（ISO26262-8/11.4.4.2d） 48
7.2.11 既知の問題とその回避策に関する記述（ISO26262-8/11.4.4.2e） 48
7.2.12 誤動作や不正な出力を検出するための方策（ISO26262-8/11.4.4.2f） 50
7.3 RTA-OSのツールインパクト（TI） 50
7.4 RTA-OSのツールエラー検出（TD） 51
7.5 RTA-OSのツール信頼度（TCL） 51

7.6 ツール認定 52
7.6.1 使用実績に基づく信頼性向上 52
7.6.2 開発プロセスの評価 54
7.6.3 ソフトウェアツールの評価 56
7.6.4 安全規格に従った開発 56

第3部 RTA-OSで生成したコードの使用に関する考察 57

8 組み込みソフトウェアの文脈におけるRTA-OS 58

8.1.1 RTA-OSによるコード生成 58

9 RTA-OSにより生成されたコードに関する想定集 60

9.1 RTA-OSにより生成されたソースコードの使用に関する想定 60
9.2 RTA-OSの実装に関する想定 67

10 生成されたRTA-OSコードの安全性を証明する議論 69

10.1 故障モード 69
10.2 開発プロセス 71
10.3 RTA-OSにより生成されたコードの安全性に関与する特性 71

11 お問い合わせ先 72

11.1 テクニカルサポート 72

12 参考文書 73

第4部（付録）RTA-OSの統合に関するチェックリスト 74

13 付録の概要 75

13.1 はじめに 75
13.2 チェックリストの使い方 75

14 RTA-OSツールの統合に関するチェックリスト 76

15 生成されたRTA-OSコードの統合に関するチェックリスト 78

16 プロジェクト情報のテンプレート 95

図

図 2-1: ISO26262のソフトウェア開発ライフサイクル 15
図 2-2: RTA-OS に対するISO26262認証の適用範囲.....16
図 3-1: ISO26262-6のソフトウェア開発プロセス.....17
図 3-2: RTA-OSのコンフィギュレーションとコード生成のワークフロー.....18
図 3-3: RTA-OSの一般的な処理 19
図 4-1: RTA-OSを用いたテスト駆動開発（test-driven development） 23
図 6-1: RTA-OSの開発と試験における要件追跡 39
図 7-1: ツール信頼プロセス 41
図 7-2: ツール信頼度の決定 44
図 7-3: RTA-OSのユーザーマニュアルの保存場所 47
図 7-4: RTA-OSターゲットポートのユーザーマニュアルの保存場所 47
図 7-5: ETASダウンロードセンターからKIRをダウンロード 49

表

表 6-1: RTA-OSツールの故障モードの検出に関する提案事項 37
表 7-1: ツール評価要件.....44
表 7-2: RTA-OSコード生成時に考慮されるエラー 51
表 7-3: RTA-OSのツール信頼度の決定.....51
表 7-4: TCL2のための認定メソッド 52
表 7-5: TCL3のための認定メソッド 52
表 7-6: 使用実績に基づく信頼のエビデンス要件 53
表 7-7: 使用実績に基づく信頼の情報要件 54
表 7-8: 開発プロセス評価の要件 54
表 7-9: 開発プロセスの評価の要件.....55
表 7-10: ソフトウェアツールの要件に対する妥当性確認とエビデンス.....56
表 10-1: RTA-OSツールの故障を検出するための提案 70

第 1 部 概要

1 概要

本書は、安全に関連するシステムにおいて ETAS の RTA-OS 製品を使用するためのガイドブックで、一般情報、および ISO26262 に特定した情報がまとめられています。ISO26262 の概要と、それを開発に適用しながら RTA-OS を使用する方法は、第 2 部に説明されています。

本書には、システムの「**セーフティケース**（**safety case**）」（システムが安全であるということに関しての、完全で一貫性した包括的かつ正当な「議論（argument）」）の一部として使用できる基本情報が示されており、以下のトピックで構成されています。

- RTA-OS がシステム開発のどのような**文脈**（**context**）において使用されるか
- RTA-OS の安全性が構築された根拠に対する安全コンセプトの**想定**（**assumption**）
  - ユーザーは、これらの想定について妥当性を確認する必要があります。
- RTA-OS の**安全性を証明する議論**（**safety argument**）
  - システム全体の安全性に関与する RTA-OS **製品**（**product**）に固有なプロパティについての根拠
  - RTA-OS の開発に用いられた**プロセス**（**process**）についての根拠
- 安全性を証明する議論を支持する**エビデンス**（**evidence**: 証拠）の入手先
  - ETAS は、ご要望に応じて、エビデンスとして使用できるもの（設計時の中間生成物、トレースエビデンス、レビューと試験の手順書、稼働統計など）をご提供いたします。
  - 開発時において、ソフトウェア製造者はいくつかのエビデンスを生成する必要があります。
- RTA-OS を使用したシステムの安全性を確保するため、開発時の別の工程でソフトウェア製造者が果たすべき**責務**（**obligation**）

1.1 本書の対象ユーザー

本書は、安全性の実現にソフトウェアが重要な役割を果たすシステムの設計と実装に関わる専門家を対象としています。構築しようとするシステムの故障モードについての詳細な知識が必要で、システムレベルの危険分析を行って安全要件と安全コンセプトを導出する方、または、ソフトウェアで実現されたシステムのセーフティケースの記述または技術的レビューを行う方が対象となります。

特に、RTA-OS が正しく機能することによりどの安全要件が満足されるかを把握している必要があります。それにより、システム全体の安全性に関連して RTA-OS を使用する文脈が決まります。

RTA-OS のユーザー（以下「ユーザー」）には、以下についての知識や経験が必要です。

- AUTOSAR のアーキテクチャ
- C 言語による組み込みプログラミング
- ソフトウェアエンジニアリングの業務とその業務に使用するツール
- ISO26262:2011、IEC61508:2010、またはそれに類する規格（EN50128 など）

1.2 本書の使い方

RTA-OS ツールを使用して安全性に関わる ECU システム用の AUTOSAR オペレーティングシステムを生成する際には、本書をよく読んで以下のことを行ってください。

1. RTA-OS ツールとそれにより生成されたコードが、ECU 開発プロセスにどのように適

用される（または適用できる）かを理解する

2. 開発工程において RTA-OS ツールをどのように使用するかを計画する
3. 生成された RTA-OS コードをどのように ECU アプリケーションに統合するかを計画する
4. 推奨される最良の方法で RTA-OS が使用されることを確認する

本書の第 1 部では、RTA-OS 製品について、その背景と ECU 開発における役割を示し、さらに ISO26262 規格の関連する部分について概説します。

本書の第 2 部では、ECU 開発工程においてどのように RTA-OS ツールを使用すればよいかを詳しく説明します。

本書の第 3 部では、RTA-OS の生成済みコードを ECU アプリケーションにインクルードする方法を詳しく説明します。

第 2 部と第 3 部には RTA-OS ツールや生成済みコードに関するさまざまな「前提条件」が述べられています。各前提条件が実際の環境に該当するかどうかはユーザーが判断する必要があり、該当する前提条件については、ユーザーの責任において RTA-OS を正しく使用する必要があります。多くのケースについては、責務を実行するために推奨される方策が提示されています。

各前提条件について、ユーザーの状況に該当するかどうかが不明の場合は、ETAS のサポート窓口までお問い合わせください。ETAS は、安全関連のソフトウェア開発のあらゆる面をカバーするコンサルティングを行っています。お問い合わせ先は第 11 章を参照してください。

本書の第 4 部に掲載されているチェックリストは、実装レビュー工程、つまり RTA-OS ツールが正しく使用され、生成された RTA-OS コードがアプリケーションに正しく統合されたかを実証する工程において利用できるものです。末尾には、レビュー対象の ECU 環境についての詳しい情報を記入するためのプロジェクト情報テンプレートが用意されています。

1.3 安全に関する注意事項

警告！

本製品の誤った使用は、死亡、または重傷を負うなどの危険を伴う場合がありますので、ご使用になる前に必ず下記の事項およびユーザーマニュアルの記載事項をよくお読みいただき、必ずその指示に従ってください。

ETAS 製品は標準的な品質管理要件を満たしています。特定の安全基準（IEC 61508、ISO 26262 など）への対応が必要な場合は、ETAS に対してその要件を明示的にご提示いただき、ご依頼いただく必要があります。またその場合は、製品のご使用開始前に所定の安全基準に準拠しているかをユーザー側でご検証いただく必要があります。

本製品を使用することにより、車両またはテストベンチの電子システムに影響を与えたり、同システムに対する制御を行ったりすることが可能となります。そのため本製品は、本製品および本製品の使用対象となる製品（試作品などを含む）に関する専門のトレーニングを受けた技術者や専門的な知識および経験を有する人が使用することを前提として設計されています。

本製品を不適切に使用した場合や、専門的な知識なしに本製品を使用した場合、車両またはシステムの挙動に影響を及ぼし、死亡や重傷などの人身傷害のほか財物の損壊を引き起こす危険性がありますので次の各項目を遵守してください。

- 操作するための十分な専門のトレーニングを受けていない場合は、本製品を使用しないでください

- 本製品に問題が生じた場合、ETAS はその問題に関する報告書（KIR: Known Issue Report、以下「KIR」と記します）を作成し、弊社ホームページに掲載します。KIR には、技術的な影響およびその解決策に関する情報が含まれていますので、ユーザーは本製品を使用する前に必ず本製品バージョンに関わる当該報告書の関連する指示をご一読の上、必要に応じて適切な対応を取る必要があります。KIR は以下の URL からダウンロードできます。

  http://www.etas.com/kir

- ユーザーが本製品によって得た一切のデータは、ユーザーが使用や配布を行う前に信頼性、品質および精度についてユーザーの責任において検証を行う必要があります。このことは、本製品の使用対象となる製品（試作品などを含む）の適合作業において基礎として使用される適合データや測定データのいずれに対しても適用されます。
- 車両の挙動に影響を及ぼし安全性に影響を与え得る車両の電子システムとともに本製品を使用する場合は、車両自体やその電子システムなどに故障や危険な状態が発生したときに車両が安全な状態に復帰できるようになっていることを必ず確認してください。（例えば、緊急停止システムが正常に作動することを確認ください。）
- テストベンチの挙動に影響を及ぼし安全性に影響を与え得るテストベンチの電子システムとともに本製品を使用する場合は、テストベンチ自体やその電子システムなどに故障や危険な状態が発生したときにテストベンチが安全な状態に復帰できるようになっていることを必ず確認してください。（例えば、緊急停止システムが正常に作動することを確認ください。）
- 本製品を使用する際には、車両およびテストベンチに関する規制や法律を含む全ての法的要件を完全に満たしている必要があります。
- 本製品を車両に搭載して使用する場合は、周囲がフェンス等で囲まれているテストコースを使用してください。

  本製品を公道において使用する場合は、所定の適合と設定を事前に検査の上、安全を確認するとともに、必要に応じて関係官庁の許可を得てください。

警告！

上記の各項目に従わない場合、死亡や重大な傷害のほか財物の損壊を引き起こす危険性があります。

ETAS グループに属する企業、その各事業所、代理店および子会社は、ETAS 製以外のソフトウェアおよびモデルコンポーネントが ETAS 製品とともに使用され、あるいは ETAS 製品にアクセスするために用いられた場合、ETAS 製品の適合性、性能および安全性に関する機能的不具合についての一切の責任を負いません。また、ETAS は、ETAS 製以外のソフトウェアおよびモデルコンポーネントが ETAS 製品とともに使用され、あるいは ETAS 製品にアクセスするために用いられた場合、ETAS 製品の商品適格性や適合性についていかなる保証も行いません。

ETAS グループに属する企業、その各事業所、代理店、および子会社は、本製品の不適切な使用によって生じた人的・物的損害について一切の責任を負わないものとします。なお、ETAS はユーザーに対して本製品を適切に使用するためのトレーニングプログラムを提供します。

1.4 用語と略語

本書では以下の用語や略語が使用されています。本書（日本語版）では、各用語の「意味」の欄に付記された日本語の用語も併用しています。

用語／略語	意味
API	Application Programming Interface - アプリケーションプログラミングインターフェース

ASIL	Automotive Safety Integration Level - 自動車向け安全度水準
ARXML	AUTOSAR XML configuration language （参考文書 **[2]**） - AUTOSAR XML 構成言語
BSW	AUTOSAR Basic Software - AUTOSAR 基本ソフトウェア
COM	AUTOSAR Communication - AUTOSAR 通信
DMA	Deadline Monotonic Analysis - デッドラインモノトニック分析
ECC	Extended Conformance Class - 拡張コンフォーマンスクラス
FEWI	Failure, Error, Warning or Information, generally relating to a message reported by RTA-OS to the user. - 故障、エラー、警告、情報（RTA-OS がユーザーに対して通知するメッセージ）の総称
IEC61508	IEC61508:2010 – IEC61508 規格の第 2 版 - IEC61508-Part/Section に示される各パート／セクションへの参照
ISO26262	ISO26262:2011 – ISO26262 規格（参考文書 **[4]**）の第 1 版 - ISO26262-Part/Section に示される各パート／セクションへの参照
ISR	Interrupt Service Routine or Interrupt Sub-Routine - 割り込みサービスルーチン、または割り込みサブルーチン
OS	AUTOSAR Operating System （参考文書 **[1]**） - AUTOSAR オペレーティングシステム
OSEK	The OSEK/VDX operating system （参考文書 **[6]**） - OSEK/VDX オペレーティングシステム
MISRA	Motor Industry Software Reliability Association
MPU	Memory Protection Unit - メモリ保護ユニット
MMU	Memory Management Unit - メモリ管理ユニット
NDA	Non-Disclosure Agreement - 機密保持契約
RTE	AUTOSAR Run-Time Environment - AUTOSAR ランタイム環境
SC1	AUTOSAR OS Scalability Class1（AUTOSAR OS スケーラビリティクラス 1）: 保護機能が含まれないコア OS 機能のグループ
SC2	AUTOSAR OS Scalability Class2（AUTOSAR OS スケーラビリティクラス 2）: タイミング保護機能が含まれるコア OS 機能のグループ
SC3	AUTOSAR OS Scalability Class3（AUTOSAR OS スケーラビリティクラス 3）: メモリ保護機能が含まれるコア OS 機能のグループ
SC4	AUTOSAR OS Scalability Class4（AUTOSAR OS スケーラビリティクラス 4）: タイミング保護とメモリ保護の機能が含まれるコア OS 機能のグループ
SWC	AUTOSAR Software Component - AUTOSAR ソフトウェアコンポーネント

TCL	Tool Confidence Level - ツール信頼度
TD	Tool Error Detection - ツールエラー検知
TI	Tool Impact - ツールインパクト （ツールの不具合がシステム全体に与える影響）
UKS	RTA-OS Unified Kernel Specification Document - RTA-OS 統一カーネル仕様ドキュメント
WCET	Worst case execution time - ワーストケース実行時間
XML	eXtensible Markup Language - 拡張可能マークアップ言語

1.5 表記上の規則

本書は以下の規則に従って表記されています。

`OCI_CANTxMessage msg0 =`	コードは等間隔フォントで表記します。 各コマンドの意味や使用法はコメントとして記述され、コメントには一般的なコメント用構文が用いられています。
File → **Open** を選択して、…	メニューコマンドやフィールド名、オプション名などは**太字**で表記します。
OK をクリックして …	ボタン名は**太字**で表記します。
<ENTER> を押します。	キーボードコマンドは、**<>** で囲んで表記します。
"Open File" ダイアログボックスが開きます。	プログラムウィンドウ、ダイアログボックスなどのタイトル、およびタブ名などは、引用符で囲んで表記します。
ファイル `setup.exe` を選択します。	ドロップダウンリスト、プログラムコード、パス名、ファイル名は等間隔フォントで表記します。
「ディストリビューション」は、サンプルポイントが定義された一次元テーブルです。	強調箇所や新出用語は、「　」で囲んで表記します。

警告マークは、特に注意が必要な個所を示すものです。

親指マークは、ヒントや活用事例を示すものです。

2 ISO26262 の概要

ISO26262 は IEC61508 規格の自動車業界向けバージョンです。ISO26262 の適用範囲は、重量 3.5 トン未満の乗用車のための安全関連システムの開発です。

「システム」は、「設計に基づいて相互に関連し合う要素群で、これには少なくともセンサ、コントローラ、アクチュエータが含まれる」（ISO26262-1/124）として定義されています。

システムは階層構造であり、あるシステムが他のシステムを内包することができます。各システムには「コンポーネント、ハードウェア、ソフトウェア、ハードウェア部品、ソフトウェアユニット（ソフトウェア単体部品）」（ISO26262-1/32）が含まれます。

2.1 構成

この規格は、以下のような 10 のパートに分かれています。

1. 用語
2. 機能安全の管理
3. コンセプト段階
4. 製品開発: システムレベル
5. 製品開発: ハードウェアレベル
6. 製品開発: ソフトウェアレベル
7. 生産と運用
8. 支援工程
9. ASIL 指向と安全指向の分析
10. ISO26262 のガイドライン

パート 1～9 には各種基準が定められていて、システムはこれらの要件を満足する必要があります。パート 10 は参考情報で、例やガイドラインが含まれていますが、その内容はシステムが ISO26262 に準拠していることを宣言する際に拘束力を持つものではありません。

2.2 ソフトウェア開発への適用性

ISO26262-6/5-11 はソフトウェアレベルの製品開発に関する要件を定めていて、ソフトウェア開発ライフサイクルの以下のステップを対象としています。

- ソフトウェアレベルの製品開発の開始
- ソフトウェア安全要件の詳細仕様
- ソフトウェアアーキテクチャ設計
- ソフトウェアユニットの設計と実装
- ソフトウェアユニットの試験
- ソフトウェアの統合と試験
- ソフトウェア安全要件の妥当性確認

ソフトウェア開発要件は、RTA-OS コードジェネレータやその他の開発ツールの開発に対して直接適用されるものではありませんが、以下のいずれかを明らかにすることができます。

- これらの要件に基づいて、RTA-OS コードジェネレータのプロパティに関する要件がどのように導き出されるか（ETAS は、それらの要件を満足していることを一部またはすべてのエビデンスで示す必要があります）
- 開発プロセスで RTA-OS コードジェネレータを使用することにより、どのようにしてこれらの要件を満足できるか

これらの側面については第 3 章で説明します。

2.3 ツールへの適用

「ツール」は ISO26262 で定義されている「システム」には該当しないため、ツールに対してこの規格は適用されません。

しかしこの規格は、システム開発におけるツールの役割を認めていて、ISO26262-8/11 はユーザーが使用するツールチェーンにおけるツール認定要件を定めています。

「ツール認定」の目的は、ツールチェーン全体のセーフティケースを確立することです。従って、ツールチェーン内のすべてのツールの適格性を評価することは、「システムインテグレータ」であるユーザーの責務です。

この「認定」においては、ツールチェーンのセーフティケースを確認するため、ツールチェーン内で妥当性が未確認であるツールのうち、どのツールが正しく機能することを確認するべきであるかを明らかにする必要があります。これにより、そのようなツールにより発生した個々の障害点をなくすための確認事項を追加し、方策を盛り込むことが可能になります。

ISO26262 はツールの種類の区別をしていません。認定の観点からするとすべてのツールは同等と考えられています。

7.6 項では、認定方法についてさらに詳しく説明し、さらに、ISO26262 の要件を満足していることを示すエビデンスを RTA-OS 内で見つける方法について説明しています。

2.4 ISO26262 と RTA-OS との関係

本書では、RTA-OS コードジェネレータを ISO26262-6/5-10 と ISO26262-8/11 に則って使用する方法を説明します。図 2-1 は ISO26262-2/Introduction から引用した V モデルで、この中に RTA-OS の適用範囲（図中の青色の網掛け部）を示しています。

RTA-OS コードジェネレータは、システムアーキテクトであるツールユーザーが定義する OS についての設定と実装にのみ関わるものです。つまり、ECU システム開発の一環として ISO26262 が要求している、生成済み OS コードに関するすべての要件管理、アーキテクチャ設計、妥当性確認は、すべてユーザーが遂行しなければなりません。

アーキテクチャ要件は AUTOSAR の OS 仕様**[1]**に定義されていますが、生成される RTA-OS コードの AUTOSAR OS 仕様への適合性は、RTA-OS ジェネレータツールの ISO26262 認証によって保証されるものではありません。

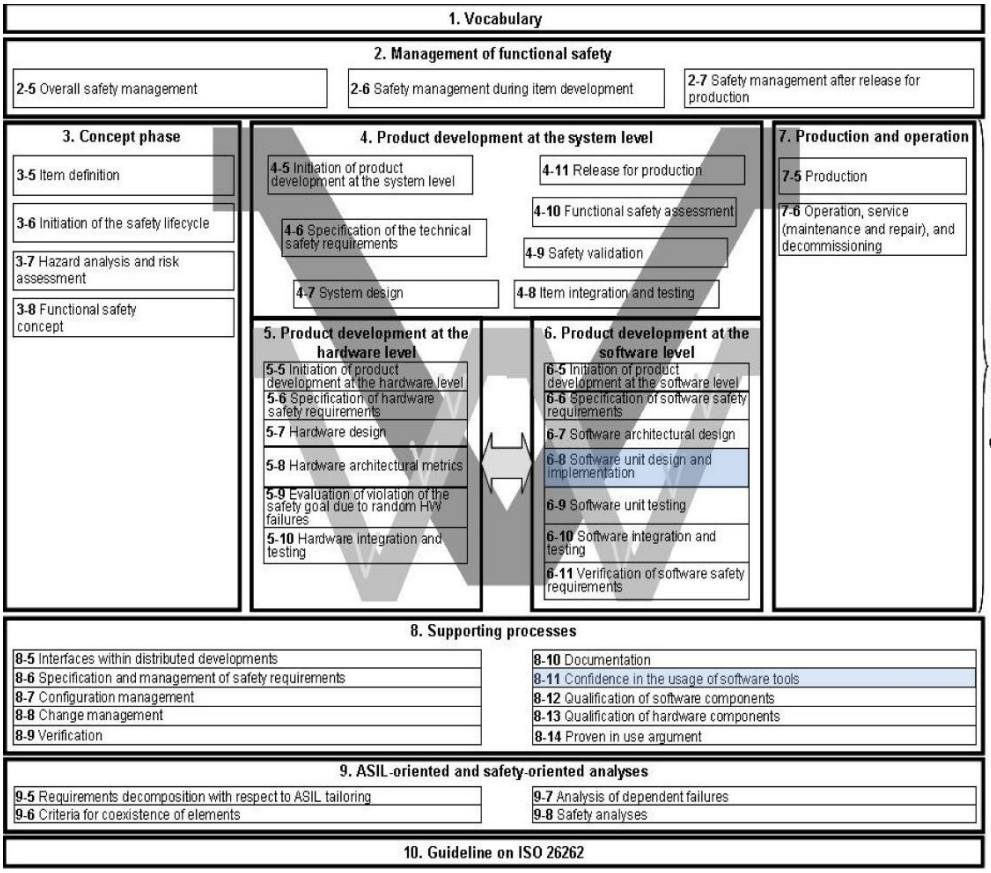

図 2-1: ISO26262 のソフトウェア開発ライフサイクル

2.5 RTA-OS ISO26262 認証の適用範囲

RTA-OS は図 2-2 に示すようなコンポーネントで構成されています。各コンポーネントは以下の 3 つのワークフローにおいて使用されます。

1. コンフィギュレーション（Configuration）: RTA-OS コンフィギュレータ（rtaoscfg.exe）を使用して、OS コンフィギュレーションの AUTOSAR XML ディスクリプションを作成します。

2. コード生成（Code Generation）: RTA-OS コードジェネレータ（rtaosgen.exe およびターゲット DLL）を使用して、AUTOSAR XML コンフィギュレーションを読み取り、OS を実装するための C ソースコードを生成します。

3. スケジューラビリティ分析（Schedualability Analysis）: RTA-OS 分析ビジュアライザ（rtaosanvis.exe）を使用して、OS のモデルに基づくスケジューラビリティ分析を行います。

上記のワークフローのうち、RTA-OS の ISO26262 認証の適用範囲に含まれるのはコード生成のみで、コンフィギュレーションとスケジューラビリティ分析は含まれません。また、RTA-OS コード生成ツールとそれに対応するターゲットポートは ISO26262 認証の範囲内ですが、生成された OS コード自体は、安全関連 ECU システムの認証の一環として扱われることになっているため、ISO26262 認証の範囲外です。従ってユーザーは、生成された RTA-OS コードを、ECU システム用にユーザーが選択した ASIL の要件に従って取り扱う必要があります。

また、生成された RTA-OS コードの AUTOSAR OS 仕様への適合性も、ISO26262 認証の範囲外です。

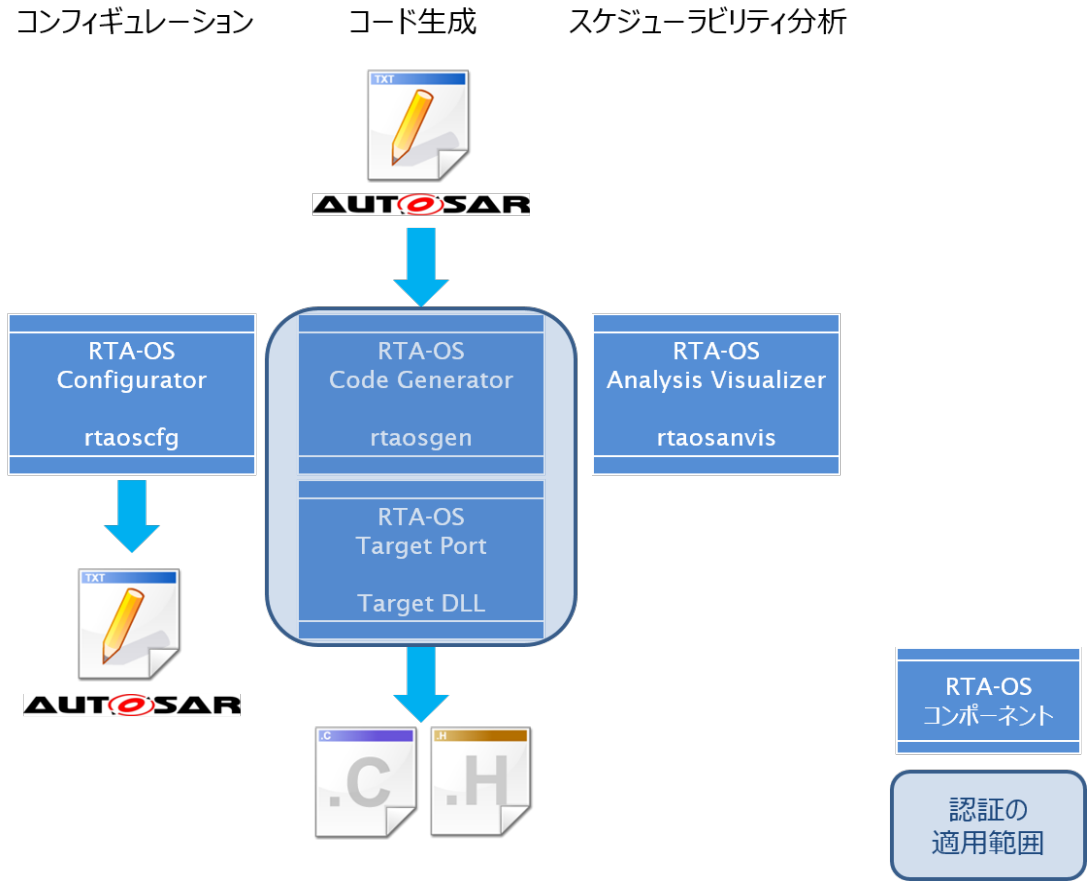

図 2-2: RTA-OS に対する ISO26262 認証の適用範囲

3 安全関連システム開発における RTA-OS の役割

RTA-OS はコマンドライン形式の OS コード生成ツール [1] です。XML 形式のコンフィギュレーションファイルを読み取って、AUTOSAR オープン規格で定義されたソフトウェアアーキテクチャを使用した自動車 ECU アプリケーションに統合される C ソースコードを生成します。

RTA-OS の処理がエラーなしで正常終了すると、AUTOSAR SWS OS 仕様の設計仕様に適合する AUTOSAR OS を実装するためのコードが生成されます。

図 3-1 は ISO26262-6/5.4 からの引用で、ISO26262 のソフトウェアライフサイクルを示しています。背景が灰色になっている部分が、このライフサイクルの中で RTA-OS コード生成ワークフローに関連するフェーズです。

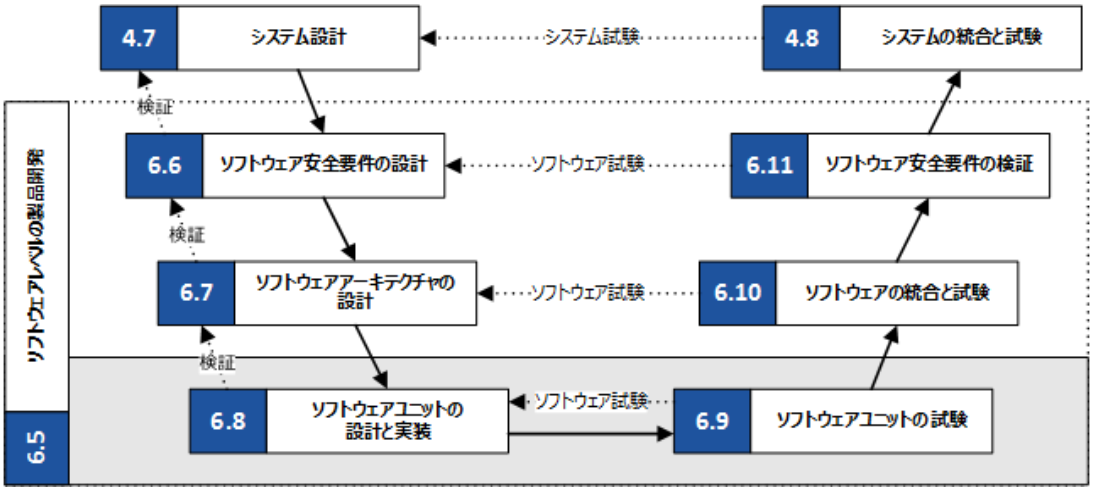

図 3-1: ISO26262-6 のソフトウェア開発プロセス

RTA-OS を使用した AUTOSAR OS ベースの開発には、「OS コンフィギュレーション」と「OS コード生成」という 2 つの主要ステップがあります。「OS コンフィギュレーション」は、一般的にはアーキテクチャ設計と詳細設計のステップにおいて決定されます。つまり OS コンフィギュレーションは、ECU システムアーキテクチャに従って設定されるか、またはチェーン内の上位ツール（RTE ジェネレータなど）により自動的に生成されます。もうひとつの「OS コード生成」は、実装工程を自動化するものです。

図 3-2 は、RTA-OS を使用した典型的な開発ワークフローを示しています。ステップの遷移を黒い実線で示し、検証作業を黒い破線で示しています。

1 ARXMLファイルの作成とコマンドライン形式のコードジェネレータの駆動に使用できるグラフィカルフロントエンドを搭載しています。

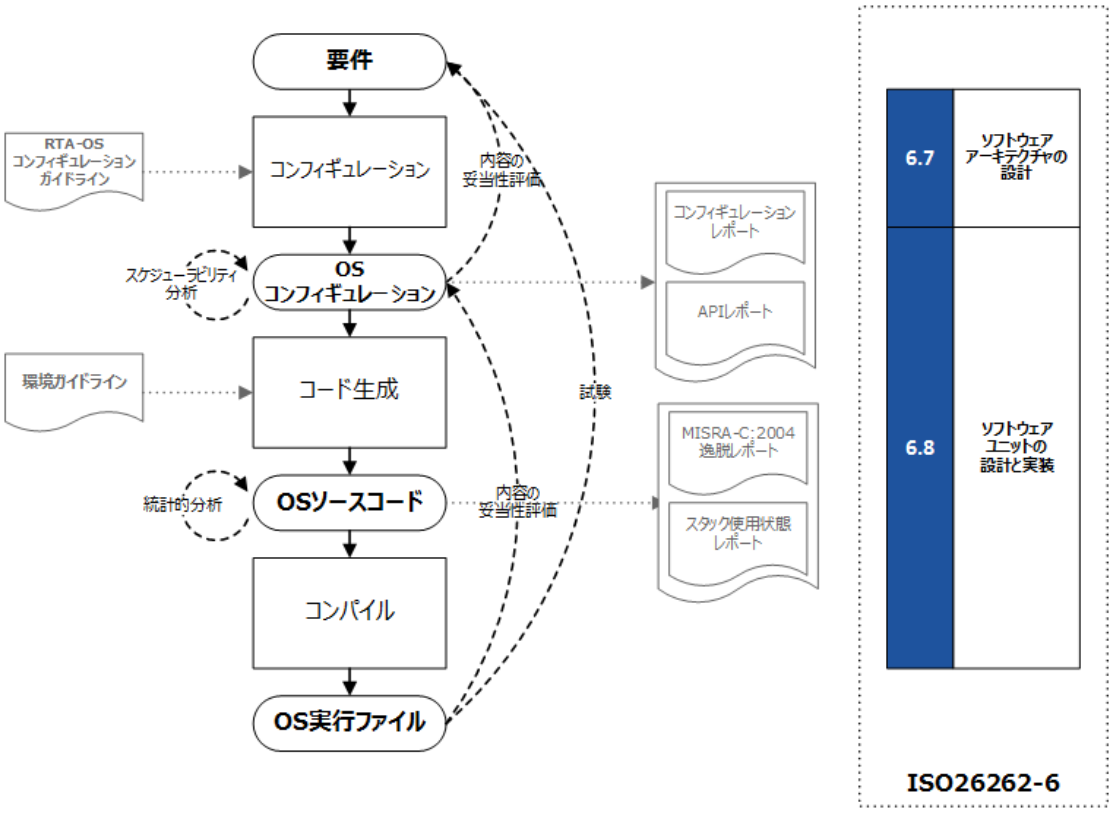

図 3-2: RTA-OS のコンフィギュレーションとコード生成のワークフロー

このワークフローの左側は、各ステップに影響を与える入力を示しています。右側は各ステップで生成される中間生成物を示し、そのさらに右側に、各ステップが ISO26262-6 の開発プロセスのどのステップに相当するかが示されています。

以降の項では、RTA-OS が行う処理の概要を示します。これは、ワークフロー内の以下の 3 つの主要ステップにおける「安全性を証明する議論」（safety argument）のためのエビデンスを確立するのに役立ちます。

- コンフィギュレーション
- コード生成
- 試験

3.1 RTA-OS の処理の概要

RTA-OS は、コンフィギュレーションのディスクリプションに基づいて AUTOSAR オペレーティングシステム（OS）を生成するためのツールです。ディスクリプションは AUTOSAR XML 形式で提供されるもので（参考文書**[2]**）、AUTOSAR オーサリングツールまたは RTA-OS コンフィギュレーションツールにより作成することができます。これらのツールはコンフィギュレーションファイルを読み取り、OS オブジェクト（タスク、ISR、カウンタ、アラームなど）のテンプレートをベースとするメカニズムを使用して、OS を構成する C コードファイルを生成します。一般的に、生成された C コードファイルはコンパイルされて実行可能ライブラリとなり、ECU システムコードにリンクされます。RTA-OS の処理の概要を図 3-3 に示します。

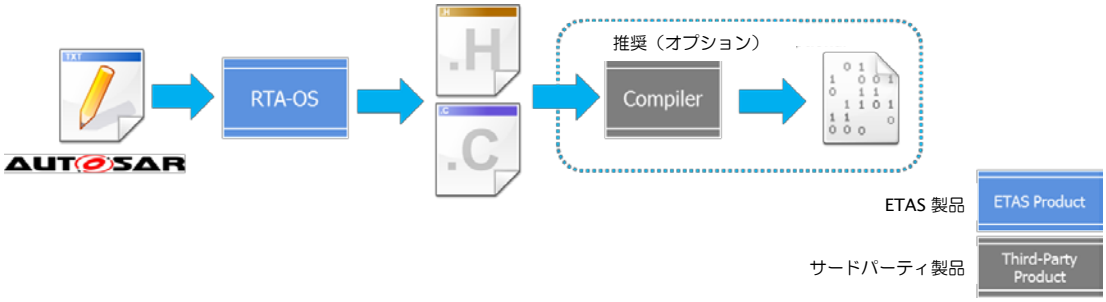

図 3-3: RTA-OS の一般的な処理

ECU システムは一般的に、マイクロコントローラの管理下で実行されるソフトウェアモジュール群で構成されます。RTA-OS はシステム内の他のソフトウェアモジュールとのインターフェースとマイクロコントローラ自体とのインターフェースを備えています。RTA-OS が他のソフトウェアモジュールに提供する API については、『RTA-OS リファレンスガイド（RTA-OS Reference Guide）』（参考文書**[9]**）で詳しく説明しています。

マイクロコントローラハードウェアとのインターフェースも、ECU システムから提供されるべきコールバック関数として『RTA-OS リファレンスガイド（RTA-OS Reference Guide）』に掲載されています。このようにして RTA-OS はマイクロコントローラハードウェアの機能（タイマ、通信チャンネル、メモリ保護ユニットなど）を一切予約しないようになっています。ただし、OS オブジェクトについて正しいスケジューリングを行うため、割り込み制御レジスタは RTA-OS の排他的制御下にあるものと想定されます。

RTA-OSは以下のものを入力として使用します。

- XML 入力ファイル
- コンフィギュレーションオプション

RTA-OSは以下のものを生成します。

- XML コンフィギュレーションに基づいて AUTOSAR OS を実装するための C オブジェクトコードライブラリ
- OS とのインターフェースが定義された C ヘッダファイル
- OS の API 関数が実装された C ソースコードファイル
- 各 API 関数を構成するコードの MD5 チェックサムを含むシグネチャファイル（これにより、ライブラリの再ビルド時に、変更されていない API 関数の再コンパイルを回避できます）
- コンソールに出力される FEWI（障害、エラー、警告、情報）メッセージ

AUTOSAR規格は以下の内容を定めています。

- XML 入力ファイルのフォーマット
- API と型宣言
- 生成される C ヘッダファイルのファイル構造
- OS が生成したコードと他の基本ソフトウェアモジュールとの間のインターフェース

3.2 ISO26262 と RTA-OS の関連性

RTA-OS: ISO26262-8 に関連する**ツール**です。ユーザーの開発プロセス内のツールの運用には、ISO26262-8/11 の要件が適用されます。

生成される OS: ISO26262-6 に関連する**ユニット**（単体部品）です。このユニットは自動的にシステムごとに生成され、生成されたコードには ISO26262-6/8 が適用されます。

警告！

RTA-OS で生成された OS コードは RTA-OS ツールの ISO26262 認証の範囲外です（2.5 項を参照）。従ってユーザーは、ユーザーの RTA-OS コンフィギュレーションに従って生成されたコードが ISO26262 の関連要件に準拠していることを、ユーザー自身の責務において、ユーザーシステム内の他のコードの場合と同様の手順、および本書に掲載されている推奨事項と提案事項を用いて実証する必要があります。

生成された OS は一般的に、ISO26262-8/12 に定義されている「ソフトウェアコンポーネント」ではありません。OS コードはプロジェクトごとに生成され、入力用 XML ファイルにより定義されるソフトウェアコンフィギュレーションの内容に完全に依存します。

第 2 部　RTA-OS ツールの使用に関する考察

4 開発プロセスにおける RTA-OS

RTA-OS は、XML 入力コンフィギュレーションの構文チェック（XML スキーマ評価）とセマンティクスチェック（ツールの一部としてコーディングされています）を行います。これらのチェックにより、入力されたコンフィギュレーションが AUTOSAR コンフィギュレーション言語の制約範囲内に収まる正しいものであるかどうかが明らかになります。

RTA-OS は、入力された XML 自体にユーザーの設計意図が正しく表現されていることを保証することはできません。これは、RTA-OS が「ガーベージイン／ガーベージアウト（つまり不完全な情報からは不完全な結果しか得られない）」ツールであることを意味します。コード生成時には、正しいケースを検出するため入力言語の定義内でチェックが行われますが、このチェックにおいては設計意図が反映されない可能性があります（未使用のイベントやリソースなど）。これらのチェックの結果は警告としてコンソールに報告されます。

RTA-OS が生成する C コードとアセンブラコードは、コンパイルしてアプリケーションの他の部分とリンクする必要があります。コード生成は特定のマイクロコントローラファミリとコンパイラをターゲットにして行われるため、生成されたコードを他のマイクロコントローラや他のコンパイラ、または同じコンパイラの他のバージョンにポーティングすることは考慮されていません。

RTA-OS が生成した OS コードは AUTOSAR ソフトウェアスタックに統合することができます。この統合を容易にするために、生成されたコードにおいては所定の AUTOSAR ヘッダファイル（たとえば `Compiler.h` や `Platform_Types.h` など）が ECU システムレベルで提供されるものと想定されています。そのため、RTA-OS で生成される OS とやりとりするすべての BSW モジュールが同じ AUTOSAR 標準ヘッダファイルを使用する必要があります。

RTA-OS で生成される OS コードをアプリケーションに統合する際には、要求される完全性レベルに適した試験技法とカバレッジ尺度を確立する必要があります。RTA-OS ツールは ECU システム開発のワークフロー全体を通じた試験／評価作業をサポートすることを念頭に置いて設計されているため、試験／評価結果を早期に得ることができ、OS コンフィギュレーションの正当性の確認に役立てることができます。試験は、ステップに応じて、シミュレーション、仮想開発環境、ラピッドプロトタイピング環境、ECU システムハードウェアといったさまざまな環境内で実行することができます。

警告！

RTA-OS のユーザーは、RTA-OS で生成されたコードに関して、ユーザーが選択した ASIL の要件に応じた妥当性確認を行う責任があります。この作業には、適切なレビュー、カバレッジ、試験などの技法などが含まれます。

開発サイクルの終盤に重大な問題が生じるのを防ぐには、早期の試験が役立ちます。つまり、早期に試験を行う目的は開発効率を最大限に高めることであり、後続の開発ステップを省略することではありません。

本章では、RTA-OS ツールの機能に基づく試験のワークフローを提示します。ここでは以下の作業も含まれています。

- OS コンフィギュレーションを修正する
- OS コンフィギュレーションに基づいて正しいコードが得られたことを確認する

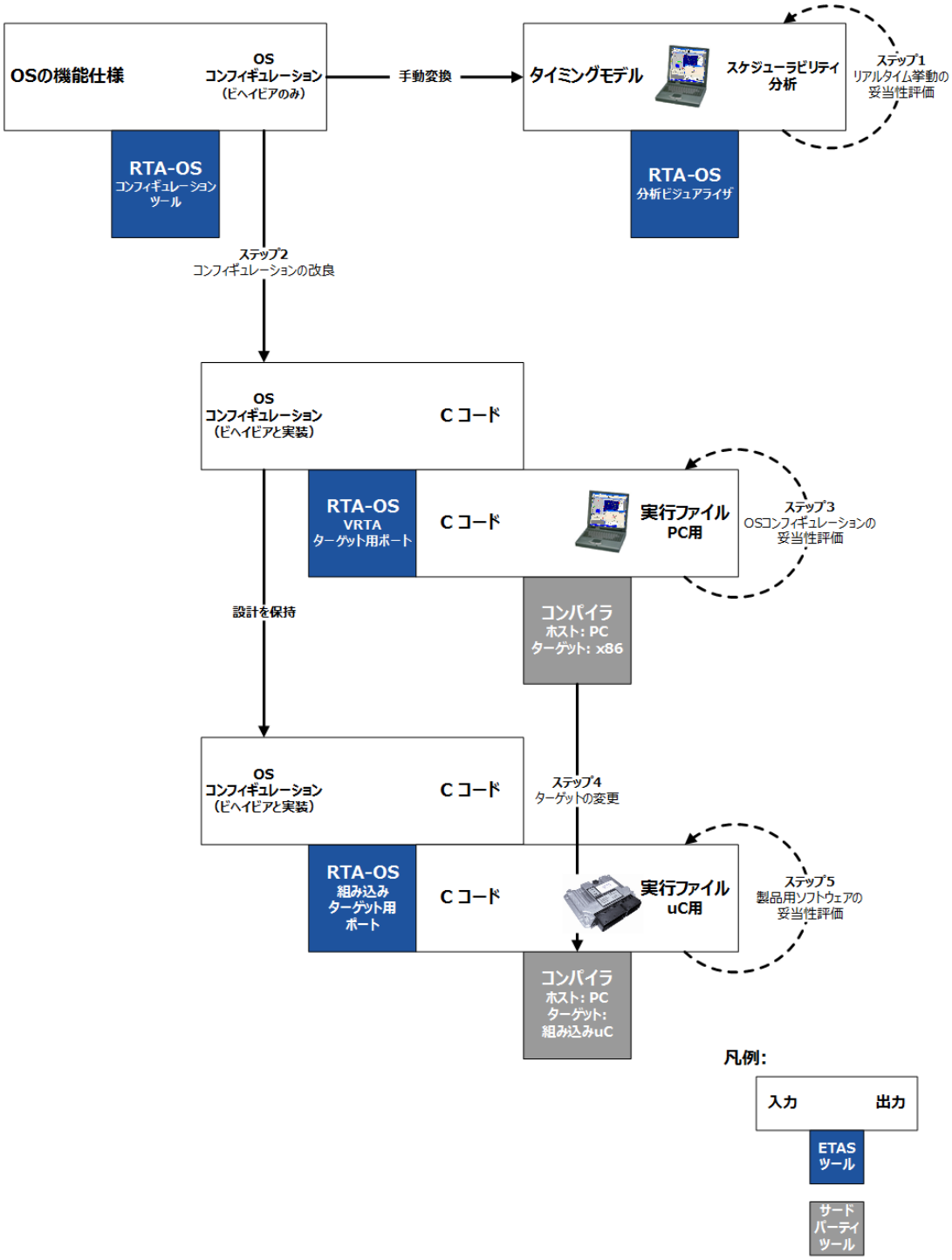

図 4-1: RTA-OS を用いたテスト駆動開発（test-driven development）

テスト駆動開発（test-driven development）

図 4-1 は、RTA-OS を用いた「テスト駆動開発」の基本戦略を示したものです。この戦略では、プロジェクトを ECU システムハードウェアに移す前に、シミュレーション、仮想開発、さらに必要に応じてラピッドプロトタイピングを利用して試験を行い、OS コンフィギュレーションの正当性を早期に確立します。この戦略は以下のステップで構成されます。

ステップ 1: OS コンフィギュレーションのリアルタイム要件を評価することを目的に、OS コンフィギュレーションから導出されるタイミングモデルを使用してスケジューラビリティ分析（4.2 項を参照）を行います。

ステップ 2: コンフィギュレーションを改良して、実装のための詳細情報（ターゲットハードウェアインターフェース）を追加します。

ステップ 3: 仮想ターゲットを使用して OS の挙動を評価します。

ステップ 4: 組み込みターゲットポートに切り替えて、ハードウェアインターフェースとタイミング挙動を仕上げます。

ステップ 5: 量産ソフトウェアを試験します。

この戦略は、優れた安全関連システム開発の原則に従っています。各ステップでシステムのある 1 つの側面だけを変更してその影響を検証し、順に次のステップへ進みます。

以降の項では、この全体的なスキームが単体試験と結合試験にどのように適用されるかを考察していきます。

単体試験と結合試験

図 4-1 は、統合された ECU システム全体の観点からの試験を示していますが、ステップ 1～3 は単体試験に対しても適用できます。

ETAS は RTA-OS の各 API と OS オブジェクトをそれぞれ個々のユニット（単体部品）として扱い、そのレベルでの徹底的な単体試験を行っています。従って、RTA-OS により生成される OS ソースコードは AUTOSAR OS 仕様（参考文書**[1]**）に準拠したものになります。

警告！

ETAS が行った RTA-OS のリリース試験とは無関係に、RTA-OS のユーザーは必ず、ECU システムについてユーザーが選択した ISO26262 ASIL の要件に応じたすべての試験を行う必要があります。

ETAS は所定のハードウェア環境で RTA-OS を試験しています。それ以外の環境（サポートされているファミリとは異なるマイクロコントローラデバイスや異なるコンパイラオプションなど）を使用する場合、ユーザーは、RTA-OS で生成された OS コードについて結合試験を行い、ECU システム環境とのインターフェースが ISO26262-6/9.4.6 に従って正しく機能することを確認してください。たとえば、アラームが必ず正しいレートで満了するように、トレースツール（ETAS の RTA-TRACE など）を使用してベースとなるタイマからの割り込みの到達レートを計測し、設定されたレートと比較する、といったことなどを行ってください。

RTA-OS で生成された C コードをそのまま保持する場合は、標準の C 言語用試験ツールを使用し、ISO26262-6/9.4.5 に従って試験カバレッジを計測してください。

Back-to-Back テスト

RTA-OS コンフィギュレーション、および ECU システムに統合される生成済みコードについて、いくつかの有用な「Back-to-Back テスト」（異なる環境下での試験結果を比較するための試験）を行うことができます。

例 1: RTA-OS コンフィギュレーションから導出されたタイミングモデルを使用して行われるスケジューラビリティ分析を、AbsInt[2] や Rapita Systems[3]などから入手可能なコード分析ツールで得られた WCET の結果と比較することができます。WCET の結果から、OS タスクの実行にかかる時間がタイミングモデルで許容された時間よりも長いことがわかった場合、それはシステムタイミングに関する想定が間違っていることを意味し、妥当性を再確認する必要があることを示すものとなります。

[2] www.absint.com
[3] www.rapitasystems.com

例 2: RTA-OS は、同じコンフィギュレーションから仮想ターゲット用コードと組み込みターゲット用コードの両方を生成することができます。この 2 つの異なるコードをそれぞれのターゲット環境において試験し、その結果、異なるコンパイラによってコードが正しく処理されたことが判明すれば、コアの OS コード生成の正当性を確証することができます。生成されるターゲットコードは環境において異なりますが、その相違点はコード全体の中では比較的小さな部分であるため、残りの大部分がコンパイラの違いに影響されないことを確認することには大きな価値があります。

コンパイラやハードウェアの相違についての対抗策

RTA-OS では、同じコンフィギュレーションから、仮想開発、ラピッドプロトタイピング、組み込みマイクロコントローラターゲットなど、多くのさまざまな実行プラットフォーム用のコードを生成することができます。つまり、生成された OS コードをさまざまなツールチェーンでコンパイルした結果を容易に比較することができます。コードが正しくコンパイルされ、ユーザーが定義した試験に合格すれば、そのコードの正当性を合理的に確信できます。

図 4-1 に示した試験戦略に従って 3 つの環境（仮想、ラピッドプロトタイピング、組み込みマイクロコントローラ）で試験を行う場合、一般的には 3 種類のハードウェアアーキテクチャと 3 種類のコンパイラツールチェーンを使用して、同じ RTA-OS コンフィギュレーションから生成されたコードを検証することになります。

ターゲットコンパイラが無効なプログラムを拒否しないように、複数の異なるターゲットに対応するタイプのコンパイラを利用することをお勧めします。

各ステップで試験を行うことは、Back-to-Back テストと同じ原則に従ってコンパイルとハードウェアの実行結果をクロスチェックするのに役立つだけでなく、RTA-OS コード生成後のステップでエラーが生じていないことのエビデンスを生成するのにも役立ちます。

さらに、コンフィギュレーションを容易に別のターゲット用に切り替えることができるので、当該コンパイラが無効な C プログラムを受け入れるかどうかを調べることもできます。

4.1 他の補助プロセスとの相互作用

本項では、安全関連システム用のソフトウェアを生産する開発プロセスで RTA-OS を使用する際の保護対策について説明します。RTA-OS は、開発中にユーザーが関わる他の作業も支援できる可能性があります。

本項で扱っているトピックは、RTA-OS の安全な使用を実証するために必須のものではありません。ここでは RTA-OS が支援できる主な開発作業が示されています。

要件管理とトレーサビリティ

RTA-OS は要件管理ツールではありませんが、ETAS は、RTA-OS の開発／試験時において詳細な要件追跡プロセスを使用しており、この追跡結果はユーザーのソフトウェア開発プロセスにも役立つ可能性があります（詳細は 6.2 項を参照してください）。

RTA-OS のすべての要件は、ETAS との NDA 契約に基づきご利用いただける Unified Kernel Specification（UKS）という文書にまとめられています。これには OSEK と AUTOSAR OS の規格に基づく要件が含まれています。各要件には一意の番号が割り当てられ、RTA-OS で生成されるソースコードをマークアップする際にこの番号が使用されます。これらの要件マークアップを用いることにより、ECU システム用に生成された OS ソースコード内に必要な AUTOSAR OS 機能が存在することを確認することができます。

さらに ETAS は、コンサルティング契約の一環として、この要件追跡マークアップを利用してすべての AUTOSAR OS 要件に関する試験カバレッジを示した試験レポートを提供することができるので、ECU システムのセーフティケースを構築する際に役立てることができます。

コンフィギュレーション管理

RTA-OS はコンフィギュレーションを XML 形式のプロジェクトファイルに格納します。このファイルをバージョン管理システム内に配置することにより、容易にプロジェクトのバージョン管理を行うことができます。

同様に、生成された OS ソースコードもバージョン管理システムに格納することができます。標準のファイル比較ユーティリティ（diff ツール）を使用して異なるバージョン間のソースコードを比較し、相違がある場合はその内容を確認することができます。

RTA-OS コンフィギュレーションとソースコードは、バージョン管理システム内に保持することをお勧めします。

4.2 RTA-OS コンフィギュレーション

RTA-OS コンフィギュレーションは、一般的には ECU システムのアーキテクチャ仕様に基づいて決定されます。RTA-OS コンフィギュレーションの作成は、RTA-OS をスタンドアロン OS として使用する場合は手作業で行いますが、AUTOSAR ツールチェーンが採用されている場合は自動的に行うことができます。どちらの場合も、下記のベストプラクティスガイドラインに従い、スケジューラビリティ分析を行って潜在的誤りを検出し、ECU システムに誤りが伝播することを防いでください。

ベストプラクティスガイドライン

RTA-OS のコンフィギュレーションに関するベストプラクティスガイドラインは、『RTA-OS 入門ガイド（RTA-OS Getting Started）』（参考文書**[8]**）、『RTA-OS リファレンスガイド（RTA-OS Reference Guide）』（参考文書**[9]**）、『RTA-OS ユーザーガイド（RTA-OS User Guide）』（参考文書**[10]**）に掲載されています。最も重要なガイドラインは、このセーフティマニュアルの第 5 章にもユーザーの「責務」として取り上げられています。ユーザーの ECU システムプロジェクト内で使用されている RTA-OS の機能が第 5 章の「責務」に該当する場合は、提案されている「方策」について十分に検討し、責務を果たす際に役立ててください。

RTA-OS ではコンフィギュレーションレポートを生成することができます。このレポートはコンフィギュレーションに含まれるすべての OS オブジェクトについての詳細情報を報告するものです。この情報を参照し、想定したオブジェクトだけがコンフィギュレーションに含まれていることを照合確認してください。RTE ジェネレータと共に AUTOSAR の厳密なコンフィギュレーションチェック機能を使用して、事前定義されたオブジェクト群のみを OS コンフィギュレーションに含めるようにしておくことは、有利性につながる可能性があります。

RTA-OS では、特定の OS コンフィギュレーション用にカスタマイズされた API レポートを生成することもできます。このレポートには、標準の『RTA-OS リファレンスガイド』（参考文書**[9]**）には記述されていない特殊な API 関数や型など、ターゲットに合わせて追加された機能についての情報がすべて含まれています。このレポートを利用して、ターゲット固有の OS 機能が ECU システムプロジェクト内で確実に正しく使用されるようにしてください。

スケジューラビリティ分析

「スケジューラビリティ分析」は、タスクと ISR の実行時間と実行周期に関する情報を利用してワーストケースの応答時間を算出する数学的技法です。これは、あるタスクがワースト

ケースにおいて、そのタスクに実装された処理のデッドラインより先に処理を終えたかどうかを調べるのに役立ちます。

AUTOSAR OS は静的に設定され、優先度シーリングプロトコルを実装しているので、スケジューラビリティ分析を行うことが可能です。ただし、使用される OS 機能やスケジューラビリティ分析ツールが使用するモデルによっては、制約が存在する場合もあります。

ECU システム内にデッドラインより前に完了しなくてはならない処理が存在する場合は、スケジューラビリティ分析を行い、RTA-OS コンフィギュレーションに起因するタイミング問題が発生しないことを確認してください。コンフィギュレーション設定の段階でこの分析を行っておけば、容易に変更が行え、ソースコード内の問題が ECU システムに影響するリスクも回避できます。

RTA-OS には Analysis Visualizer というスケジューラビリティ分析ツールが含まれており、OS コンフィギュレーションをモデリングしてスケジューラビリティ分析を行う方法が『RTA-OS Analysis Visualizer User Guide』（参考文書**[7]**）に詳しく記述されています。そのほか、モデルレベルで使用できる分析ツールが Inchron[4] と Symtavision[5]から入手可能です。

[4] www.inchron.com

[5] www.symtavision.com

5 RTA-OS ツールの使用に関する想定集

本章では、安全関連システム開発時の RTA-OS の使用に関して ETAS が「**想定**」している条件を提示します。これらの想定のうち、ユーザーの ECU システムに該当していないと見られるものがある場合は、ECU システムに変更を加えるか、ETAS に連絡して（連絡先は第 11 章を参照）内容を確認してください。

想定事項の多くには、その想定を有効にするためにユーザーが履行すべき「**責務**」が併記され、さらにユーザーが責務を履行する際に役立つ「**方策**」も提示されています。

5.1 システムレベルの想定

以下は、システム全体に関して想定される事象です。これらは ETAS が RTA-OS 環境の適用範囲を規定するためのものです。

想定 1: RTA-OS ツールにより生成された OS コードは、ISO26262 に準拠した適切な ASIL（最高で ASIL D まで）が適用された安全関連 ECU システムに統合されるものとします。

責務 1: ユーザーの ECU システムに適切な ASIL を明らかにする必要があります。

想定 2: ECU システムの開発プロセスは ISO26262 に記述されている適切な ASIL の要件を満足しているものとします。

責務 2: ユーザーの ECU システム開発プロセスでは、ユーザーが選択した ASIL に関して ISO26262 が推奨している事項に確実に従う必要があります。

想定 3: AUTOSAR SWS OS（参考文書**[1]**）に記述されている AUTOSAR OS の仕様は、安全関連アプリケーション内での使用に適しているものとします。

責務 3: AUTOSAR OS の機能の中から、ユーザーの安全関連アプリケーションでの使用に適したサブセットを特定する必要があります。

方策 3a: 入力された XML と生成された OS ソースコードとを比較する機械的チェックの手段を導入し、ユーザーが選択した機能サブセットの制限が RTA-OS への入力に確実に反映されていることを確認してください。

想定 4: 生成された OS が実行されるハードウェアは、そのハードウェアのリファレンスマニュアルとそのすべての正誤表に記述されているとおりの挙動を示すものとします。

責務 4: ユーザーが選択したハードウェアが記述されたとおりに機能することを検証する必要があります。

方策 4a: RTA-OS のリリース試験から得られた情報を活用できる場合がありますので、ETAS までお問い合わせください。

想定 5: 生成された OS が実行されるマイクロコントローラは、所定の『RTA-OS Port Guide』に記述されている RTA-OS のリリース試験に使用されたものと全く同じものであるものとします。同じファミリに属する別の派生タイプが使用されている場合は、ETAS がそれを具体的に評価し、結果を記録した互換性レポートを提供しているものとします。

責務 5: ユーザーが選択したマイクロコントローラが ETAS による RTA-OS リリース試験で使用されたものと互換であることを、シリコンベンダーに確認する必要があります。

方策 5a: ETAS に対して、生成された RTA-OS コードをユーザーのハードウェア環境と全く同じ環境で評価して報告することを依頼してください。

想定 6: 生成された OS コードは、適切な『RTA-OS Port Guide』に記述されているのと同じコンパイラバージョンとオプションを使用してコンパイルされているものとします。異なるコンパイラバージョンやオプションが使用されている場合は、それらのものが ETAS によって具体的に評価され、互換性レポートが提供されているものとします。

責務 6: コンパイラバージョンとオプションをレビューして、それらが ETAS による RTA-OS リリース試験で使用されたものと互換であることを確認する必要があります。

方策 6a: ETAS に対して、生成された RTA-OS コードをユーザーの ECU システムで使用されているものと同じコンパイラバージョンとオプションを使用して評価することを依頼してください。

方策 6b: ISO26262 に対するコンパイラツールの適格性を検討してください。

想定 7: RTA-OS が実行される ECU システムに統合されている他のすべての AUTOSAR BSW モジュールは、OS コード生成に使用される RTA-OS コンフィギュレーションを作成したときに選択されたものと同じリリースバージョンの AUTOSAR に準拠するものとします。たとえば、RTA-OS コンフィギュレーションツールで AUTOSAR R4.0.3 が選択されている場合は、他のすべての BSW モジュールも AUTOSAR R4.0.3 に準拠している必要があります。

責務 7: RTA-OS ツールでは、システムの他のすべての部分に使用されている AUTOSAR リリースと同じものを選択する必要があります。

方策 7a: RTE を含む、使用されているすべての AUTOSAR BSW モジュールをレビューし、それらがすべて同じ AUTOSAR リリースに準拠していることを確認してください。そのためには、システムに統合されている各モジュールのバージョンや、コンフィギュレーションオプションとして選択されている AUTOSAR リリースが適切であるか、などの確認が必要となる場合があります。

想定 8: RTA-OS が実行されるシステムに統合されているすべての AUTOSAR BSW モジュール（RTE を含む）は、所定の AUTOSAR SWS ドキュメントに記述されているとおりの OS インターフェースを実装しているものとします。

責務 8: ユーザーの ECU システムに使用されている他のすべての AUTOSAR BSW モジュールが所定の AUTOSAR SWS ドキュメントに準拠していることを確認する必要があります。

想定 9: RTA-OS の機能のうち、ISO26262 認証を受けた部分（2.5 項を参照）のみが使用され、さらにその部分が、このセーフティマニュアルに記述された条件に従って使用されているものとします。

責務 9: ユーザーが RTA-OS ツールを使用する際にこのセーフティマニュアルの推奨事項を遵守していることを確認する必要があります。遵守していない場合は、ISO26262 に従って RTA-OS のツール認定を実施する必要があります。

方策 9a: RTA-OS について、要求されるレベルのツール認定を ISO26262 に従って行う必要があります。これには、7.2 項に説明されている TCL 用のエビデンスを使用します。

方策 9b: ユーザーの開発環境において、RTA-OS ツールについて必要なすべての試験を行う必要があります。ツール認定の技法と、ETAS が提供できる支援的エビデンスについては 7.6 項で説明します。

方策 9c: ユーザーのシステム用に選択されている ASIL について ISO26262 に定められている方策に従って、RTA-OS ツールの出力をレビューして試験する必要があります。このレビューと試験には以下の事柄が含まれますが、それ以外のものが必要となる場合もあります。

- 生成された XML コンフィギュレーションがユーザーのシステム要件に適合していて

ユーザーの安全目標に対応していることをレビューする

- 生成された OS コードが AUTOSAR OS 仕様を適切に実装していることをレビューして試験する
- RTA-OS で生成されるレポートをレビューして予想された情報が含まれていることを確認する

5.2 RTA-OS ツールの使用に関する想定

以下は、ECU 開発プロセスにおける RTA-OS ツールの使用に関して ETAS が想定している条件です。これらの想定自体が、ユーザーが安全関連システム開発時に RTA-OS を使用するときに履行しなければならない責務の基礎になります。

コンフィギュレーション

想定 10: ユーザーは、RTA-OS のユーザー文書に記述されているコンフィギュレーションに関する助言と推奨事項を読んで理解し、それらに従ったアクションを適切な場面で実行するものとします。

責務 10: RTA-OS の機能を使用する際には、RTA-OS ユーザー文書（『RTA-OS ユーザーガイド（RTA-OS User Guide）』、『RTA-OS リファレンスガイド（RTA-OS Reference Guide）』、各ターゲットポート用の『RTA-OS Port Guide』など）に記述されている助言と推奨事項に従う必要があります。

方策 10a: システムコードをレビューして、RTA-OS の機能が一貫性を保って使用されていることを確認してください。ご不明の点は ETAS にご相談ください。

想定 11: 入力用コンフィギュレーションファイルはユーザーの安全関連アプリケーションの設計意図を正しく表現しているものとします。

責務 11: ユーザーの RTA-OS コンフィギュレーションによって ECU システムが正しく動作することを確認する必要があります。

方策 11a: ISO26262 に記述されている手順と方法を用いて、システムおよび RTA-OS コンフィギュレーションの定義を入念に行う必要があります。

想定 12: 型定義、コンパイラ、メモリ抽象化を含む標準の AUTOSAR ヘッダファイルに、ECU システムに関する正しい情報が記述されていることが、システムインテグレータによって検証されているものとします。

責務 12: RTA-OS は、コンパイラツールチェーンからの抽象化のために、AUTOSAR の標準化メカニズムを使用します。ユーザーは、以下のヘッダファイルの内容がコンパイラツールチェーン、ターゲットハードウェア、データの特性（たとえば読み取り専用データは ROM に配置され、読み書き可能データは RAM に配置されるなど）に関して正しいこと、またこれらのヘッダファイルがすべての AUTOSAR モジュールに一貫して使用されることを確認する必要があります。

- Compiler.h
- Compiler_Cfg.h
- PlatformTypes.h
- MemMap.h

方策 12a: AUTOSAR 標準ヘッダファイルの使用と内容をレビューし、それらがユーザーのシステムに適していること、また RTA-OS とやりとりする他のモジュールにも同じ標準ヘッダファイルが一貫して使用されていることを確認してください。

ライブラリビルド

想定 13: 生成された RTA-OS コードは、RTA-OS ツールによって自動的にライブラリに組み込まれます。これには、ライブラリのビルドに使用されるコンパイラオプションが 、ETAS による RTA-OS のリリース試験に使用されたものと同じになるという利点があります。

責務 13: RTA-OS ツールを ECU ビルドプロセスに統合することにより、生成された RTA-OS ライブラリが最終的なアプリケーションの実行可能ソフトウェアに確実に含まれるようにする必要があります。

想定 14: 生成された RTA-OS コードはデバッグと評価のためにのみ使用されるものとします。RTA-OS ツールと無関係にコンパイルされることはありません。

責務 14: 生成された RTA-OS コードを RTA-OS ツールとは無関係にビルドしようとする場合は、必ず、ETAS による RTA-OS リリース試験に使用されたものと同じコンパイラオプションを使用する必要があります。

方策 14a: 所定の『RTA-OS Port Guide』に記述されているコンパイラオプションをレビューして、それと同じオプションをビルドプロセスに使用してください。

方策 14b: 生成された RTA-OS コード用に別のコンパイラオプションを使用することが要求されている場合は、ETAS に対して、生成された RTA-OS コードをそれらのオプションと共に再評価することを依頼してください。

スケジューリング

想定 15: アラームとスケジュールテーブルを駆動するソフトウェアカウンタの挙動の制御に使用されるハードウェアタイマが生成するすべての‘tick’割り込みは 、RTA-OS ツールでカテゴリ 2 ISR として設定されるものとします。

責務 15: マイクロコントローラのベクタテーブル内の‘tick’割り込みエントリと同じ内容が RTA-OS に設定されていることを確認する必要があります。

方策 15a: RTA-OS コンフィギュレーションをレビューして、‘tick’割り込みが OS によって正しく認識されることを確認してください。割り込み認識ロジックにシステムが供給するデマルチプレクシングなどの修正が加えられている場合は、特にこの調査が重要です。ご不明の点は ETAS にご相談ください。

想定 16: アラームとスケジュールテーブルを駆動するハードウェアカウンタの挙動の制御に使用されハードウェアタイマが生成するすべての‘advance’割り込みは 、RTA-OS ツールでカテゴリ 2 ISR として設定されるものとします。

責務 16: マイクロコントローラベクタテーブル内の‘advance’割り込みエントリと同じ内容が RTA-OS に設定されていることを確認する必要があります。

方策 16a: RTA-OS コンフィギュレーションをレビューして、‘advance’割り込みが OS によって正しく認識されることを確認してください。この調査は、特に割り込み認識ロジックにシステムが供給するデマルチプレクシングなどの修正が加えられている場合は重要です。ご不明の点は ETAS にご相談ください。

想定 17: スケジュールテーブルの暗黙的同期化が使用されている場合は、スケジュールテーブルとそれらを駆動するために使用されているカウンタとが同じ値でラップされるものとします。

責務 17: スケジュールテーブルの期間を駆動カウンタの MAXALLOWEDVALUE+1 と同じにする必要があります。

方策 17a: RTA-OS ツールを使用して、駆動カウンタの MAXALLOWEDVALUE+1 と同じ値をスケジュールテーブルの期間として設定してください。

想定 18: スケジュールテーブルの明示的同期化が使用されている場合は、スケジュールテーブルとそれらを同期化するために使用されているカウンタとが同じ値でラップされるものとします。

責務 18: スケジュールテーブルの期間を同期化カウンタの MAXALLOWEDVALUE+1 と同じにする必要があります。

方策 18a: RTA-OS ツールを使用して、同期化カウンタの MAXALLOWEDVALUE+1 と同じ値をスケジュールテーブルの期間として設定してください。

想定 19: 拡張タスクとイベントが使用されている場合は、システム分析が行われ、拡張タスクの周期的なイベント待ちによってシステムのデッドロックが発生しないことが確認されているものとします。

責務 19: ランタイムにおいて、拡張タスクの使用によるシステムのデッドロックが発生しないことを確認する必要があります。

方策 19a: 安全関連システムには拡張タスクを使用しないでください。

方策 19b: システム内に拡張タスクが存在している場合は、その代わりに基本タスクを使用し、拡張タスクの機能を基本タスクで実現することを検討してください。ご不明の点は ETAS にご相談ください。

方策 19c: システム内で拡張タスクを使用しなければならない場合は、システムコードを入念に分析し、拡張タスクが周期的なイベント待ち状態に陥る状況が発生しないようにしてください。

想定 20: スピンロックが使用されている場合は、システム分析が行われ、互いに異なるマイクロコントローラコアで実行される複数のタスクによる周期的なスピンロック待ちによってシステムのデッドロックが発生しないことが確認されているものとします。

責務 20: ランタイムにおいて、スピンロックの使用によるシステムのデッドロックが発生しないことを確認する必要があります。

方策 20a: 1 つのシステム内ではスピンロックオブジェクトを 1 つのみ使用し、コア間ですべてのデータを整合させてください。

方策 20b: 1 つのシステム内で複数のスピンロックオブジェクトを使用しなければならない場合は、必ず SC3 または SC4 を使用し、スピンロックのネスティングに関する拡張ステータスチェックが RTA-OS によって確実に実行されるようにしてください。

方策 20c: 1 つのシステム内で複数のスピンロックオブジェクトを使用しなければならない場合は、システムを入念に分析し、タスクが周期的なスピンロック待ち状態に陥る状況が発生しないようにしてください。

タイミング保護

想定 21: ECU システム全体を保護してハングアップを防止するため、OS の外部でウォッチドッグが使用されるものとします。生成された RTA-OS コードに実装される AUTOSAR OS のタイミング保護機能は、システムのタイミング問題を引き起こす可能性のあるすべての状況をカバーできるものではありません。

責務 21: ECU システム内の、RTA-OS とは無関係の部分に、深刻なタイミング問題から回復するためのメカニズムを提供する必要があります。

方策 21a: AUTOSAR ウォッチドッグタイマ（参考文書**[3]**）を使用してタイミング違反からの回復メカニズムを実装することを検討してください。

想定 22: RTA-OS コンフィギュレーションにより実現される ECU システムのタイミングモデルは、システムのリアルタイム挙動を正確に反映するものとします。RTA-OS は、このタイミングモデルによってタスク／ISR がランタイムにおいてリアルタイムデッドラインを守ることができるかどうか、という確認はしません。ECU システムのリアルタイム性能が重要である場合は、RTA-OS のコンフィギュレーションを設定する前に、タイミングモデルの分析（タスク／ISR や DMA の WCET の見積もりなど）を行い、 RTA-OS で生成された OS コードがアプリケーションに統合された後に、観察された ECU システム性能をこの分析結果に照らして評価してください。

責務 22: ECU システム用のタイミングモデルがランタイムにおいて確実に実現されるようにする必要があります。RTA-OS は定義されたタイミングモデルをそのまま実行するだけであり、そのタイミングモデルが実際に成功するかどうかというチェックは行いません。

方策 22a: RTA-OS のコンフィギュレーションを定義する前に、タイミング分析ツールを使用して ECU システム用のタイミングモデルを検証してください。さらに、同じツールを使用して、実際に観察された ECU システムのランタイム挙動を検証してください。

想定 23: 能動的なタイミング保護機能がシステム内で使用される場合は、実行バジェットを ‘tick’の値に丸めることの効果が分析されているものとします。RTA-OS のタイミング超過検知メカニズムによる誤検知を防ぐため、実行バジェットは必要に応じて調整されます。

責務 23: RTA-OS ツールで能動的タイミング保護を設定する際には、すべての丸め処理の影響を考慮する必要がります。

メモリ保護

想定 24: すべてのトラステッドソフトウェアはマイクロコントローラの同じ特権レベルを共有するので、AUTOSAR OS は他から干渉されない FFI（”freedom from interference”）ソリューションを提供することはできません。つまり、他のトラステッドソフトウェアコンポーネントが OS 用メモリに上書きしてしまう可能性があります。

責務 24: ハードウェアとソフトウェアのすべてのシステムコンポーネントが協働して有効なソリューションを確実に提供できるようにする必要があります。また、トラステッドソフトウェアコンポーネント（AUTOSAR BSW など）を調べて、それらが干渉の問題を引き起こさないようにする必要があります。

方策 24a: ECU システムの設計を定義する際に、ISO26262-6:2011/7.4.10-13 の要件を考慮する必要があります。

想定 25: システム内でメモリ保護機能が使用されている場合は、マイクロコントローラに MPU または MMU が搭載され、メモリマップを複数の独立領域に仕切る機能が提供されているものとします。

責務 25: 使用するマイクロコントローラが、ユーザーアプリケーションに必要なメモリ保護メカニズムをサポートする MPU または MMU を提供していることを確認する必要があります。

方策 25a: ECU システムにおけるマイクロコントローラ MPU の使用について定義する際には、ISO26262-6:2011/7.4.11 の要件を考慮する必要があります。

想定 26: システム内でメモリ保護機能が使用されている場合は、マイクロコントローラが「特権モード」と「非特権モード」という動作モードを提供するものとします。

責務 26:　使用するマイクロコントローラが特権モードと非特権モードという動作モードを提供していることを確認する必要があります。

方策 26a:　ECU システムにおけるメモリ保護の使用について定義する際には、ISO26262-6:2011/7.4.11 の要件を考慮する必要があります。

想定 27:　システムに信頼されている（トラステッド）として設定されているコードは、マイクロコントローラ上では特権モードで実行され、メモリマップ全体およびペリフェラルレジスタの読み取り／書き込みを行うことができます。

責務 27:　RTA-OS ツールに「トラステッド」として設定されているすべてのコードが、間違ったメモリへの書き込みやペリフェラルレジスタの予想外の変更などによってシステムの安全問題を起こさないことを確認する必要があります。

方策 27a:　ECU システムのソフトウェアパーティショニングを定義する際には、ISO26262-6:2011/7.4.13 の要件を考慮する必要があります。

想定 28:　AUTOSAR OS と他の BSW のモジュールは常にトラステッドコードであり、マイクロコントローラ上では特権モードで実行されるものとします。

責務 28:　ECU システム内の各トラステッドコードモジュールが他のトラステッドモジュールに干渉しないことを確認する必要があります。

方策 28a:　ECU システムにおける AUTOSAR BSW の使用について定義する際には、ISO26262-6:2011/7.4.12 の要件を考慮する必要があります。

想定 29:　非トラステッドコードは、マイクロコントローラ上では常に非特権モードで実行されるものとします。

責務 29:　システム内の非トラステッドコードが、非特権モードで実行されるための制約条件を順守していることを確認する必要があります。

方策 29a:　ECU システムにおける非トラステッドコードの使用について定義する際には、ISO26262-6:2011/7.4.11 の要件を考慮する必要があります。

想定 30:　マイクロコントローラは、非特権モードで実行されるコードが MPU または MMU 制御レジスタにアクセスすることを禁止するものとします。

責務 30:　使用するマイクロコントローラが、非特権モードで実行されているコードによる MPU または MMU 制御レジスタの書き換えを妨げることを確認する必要があります。

方策 30a:　ECU システムにおいて非特権モードで実行されるコードの挙動について定義する際には、ISO26262-6:2011/7.4.11 の要件を考慮する必要があります。

6 RTA-OS ツールの安全性を証明する議論

RTA-OS ツールは、ECU アプリケーションの開発において OS コードを生成するためのツールです。システムに実装される安全性を把握するには、ツールがどのように機能するかを理解する必要があり、ECU のセーフティケースに RTA-OS ツールによる影響を含めることがユーザーの責務となります。本章には、セーフティケースを構築するのに役立つ情報がまとめられています。さらに詳しい情報が必要な場合は、ETAS までお問い合わせください。お問い合わせの方法は、第 11 章を参照してください。

6.1 RTA-OS ツールの故障モード

RTA-OS の挙動は、他の多くのツールと同様に、「ガーベージイン／ガーベージアウト」ツール、つまり不完全な情報からは不完全な結果しか得られないツールであるため、いくつかの故障モードも存在します。本項の情報は、RTA-OS ツールが異常終了したかどうか、また異常終了した場合はその理由についてチェックするのに役立ちます。

RTA-OS ツールに入力される情報には、以下のものがあります。

- ARXML 入力ファイル
- コマンドラインオプション
- AUTOSAR の標準ヘッダ
- RTA-OS のシグネチャファイル
- コンパイラバージョンとシグネチャ情報

RTA-OS の通常動作は、有効な入力を受け取って有効な OS を生成することです。

RTA-OS には以下のような故障モードがあります。

1. **無効な入力を拒否**し、OS を生成しません。
2. **無効な入力を受け入れ**、未定義の挙動を取る OS を出力します。
3. **有効な入力コンフィギュレーションを拒否**し、OS を生成しません。
4. **有効な入力を受け入れ**、未定義の挙動を取る OS を出力します。

以下の表 6-1 には、RTA-OS ツールで発生する可能性のある故障モードがまとめられています。発生確率（Probability）と影響度（Impact）の値は ETAS の経験に基づいて設定されたものであるため、ユーザーの ECU システム用に異なる値を設定することが可能です。いずれにしても、発生確率と影響度の高い故障モードを中心に試験を行ってください。

なお、故障モード 1 は RTA-OS ツールの通常動作に分類することもできますが、完全を期すためにここに記載しており、ツールの異常終了のリスクを低減するための確認方法などが示されています。

#	故障モード	考えられる故障状態	発生確率	影響度	確認の方法
1	1	入力された ARXML が、所定のスキーマに対して無効である	高	低	RTA-OS はすべての ARMXL 入力を所定のスキーマと比較し、非互換箇所があればエラーを発行します。
2	1	互いに互換でない可能性のある異なるスキーマに基づく ARXML が混在している	中	低	すべての ARXML ファイルが同じスキーマを共有していない場合、RTA-OS は入力を拒否します。OS は生成されません。入力の評価は grep や各種 XML ツールなどを使用して容易に行えます。

3	1	互換性のない組み合わせのオプションが RTA-OS のコマンドラインに入力された	中	低	多くのコマンドラインオプションは他のオプションと無関係に使用できますが、非互換の組み合わせが使用されると、RTA-OS によって拒否されます。OS は生成されません。
4	1	RTA-OS が、不適切なコンパイラを使用してコンパイルしようとしている	中	低	RTA-OS はコンパイラの名前とバージョン文字列について基本的なチェックを行い、不適切な名前やバージョンが検知された場合は処理を停止します。OS は生成されません。 複数の異なるコンパイラバージョンが同じ名前とバージョン情報を報告する場合（シングルフロントエンド・マルチバックエンドのツールチェーンの場合によくあります）は、コンパイラがエラー（「無効なオプション」など）を報告する可能性があります。 ターゲット X 用に構築された OS をターゲット Y 用に構築されたコードとリンクしようとすると（ただし X != Y）、リンカがエラーを報告します。OS は生成されません。
5	2	RTA-OS のコンパイルに使用される AUTOSAR 標準ヘッダファイルが他の BSW／ASW モジュールと一致しない	中	高	RTA-OS はこの状況を検知**できません**。コンパイがこの状況を検知し、RTA-OS がコンパイラのエラー／警告メッセージをユーザーに返します。ECU システム内のソフトウェアのコンパイルには、すべて同じヘッダファイルを使用してください（前出の**想定 12** を参照）。
6	2	RTA-OS がコンパイルもリンクも行えない無効な OS を生成する	低	低	RTA-OS はコンパイラとリンカのエラーメッセージを返すので、これをコンパイラツールチェーンで検出します。
7	2	入力が変わっても、RTA-OS が OS ライブラリを正しく再ビルドしない	低	高	RTA-OS はライブラリ内の各 API 関数用に MD5 チェックサムを生成し、すべてのチェックサムをシグネチャファイルに格納します。互いに異なるビルド用のシグネチャファイルを比較して、ライブラリに変更が加えられたかどうか調べることができます。入力を変更してもシグネチャファイルが**変更されない**場合は、RTA-OS はその変更を正しく検知できていません。

8	2, 4	RTA-OS はコンパイルとリンクを行える OS を生成するが、AUTOSAR OS が正しく実装されていない	低	高	AUTOSAR または OSEK コンソーシアムによる第三者適合性試験を実施して、実装された RTA-OS が一般的に合意された挙動を取ることを実証してください。 または、参照用 OS モデルを構築して、満足のいく結果が得られるまで試験してください。この際、参照モデルには最終システムの機能が含まれている必要があります。
9	4	RTA-OS が、定義されたコンフィギュレーションに適合しない OS（例: タスクの数が不正）を生成する	低	高	入力 ARXML ファイル内に宣言されているすべてのオブジェクトに対応するオブジェクトが、生成されたコードの中に含まれる必要があります。このことは、生成されたソースコードと生成されたレポートに照らして入力をレビューすることによりチェックできます。 以下の点についてのチェックが必要です。 • オミッションエラー（定義されているオブジェクトが生成されない） • コミッションエラー（予想外のオブジェクトが生成される） • 不正な静的データとイニシャライザ（優先度、アラーム周期と起動時間、リソース上限優先度、スケジュールテーブル遅延、イベントマスクのビット数が宣言されたイベントの数と同じ、など）
10	3	RTA-OS が、有効なコンフィギュレーションを拒否する	低	低	RTA-OS から、コンフィギュレーションに関する予想外のエラーメッセージが報告されます。OS は生成されません。 個々の ARXML 入力の妥当性をスキーマに照らしてチェックする必要があります。
11	4	RTA-OS が、コンパイラツールチェーンに対して非互換なオプションを使用している	低	低	コンパイラツールチェーンが生成したエラー／警告情報が RTA-OS から返ります。コンパイラによって禁止されていないオプションはすべて互換性のあるものと判断されます。

表 6-1: RTA-OS ツールの故障モードの検出に関する提案事項

表 6-1 に示されるように、RTA-OS ツールの「故障」のほとんどは、RTA-OS かコンパイラツールチェーンのいずれかから出力されるメッセージ、または OS が生成されないことによって検知することができます。その他の故障を検知するには、RTA-OS への入力をレビューしたり、生成された OS をコンフィギュレーションに照らしてチェックしたりする必要があります。ECU システムの統合プロセスには必ずそのようなレビューとチェックの機能を組み込んでください。

6.2 RTA-OS ツールの開発プロセス

RTA-OS は ETAS 製品エンジニアリングプロセス（PEP: Product Engeneering Process）に従って開発されています。このプロセスについては 7.6.2 項に詳しく説明されています。

ETAS PEP は古典的なウォーターフォール型開発プロセスを優れた形で実現したもので、CMMI Level 3 に準拠していることが認証されています。優れたプロセスの価値は、それに従って作成されたすべての生成物が一貫性のある最終製品にならない限りは現実的なものとは言えませんが、RTA-OS の開発においては、設計から実装、試験設計、試験結果に至るまで開発要件をトレースするツールを使用することで、確実に価値を実現しています。RTA-OS の試験を行う際には必ずこのツールによる自動試験が実行されるので、すべての要件が実装され、試験されたことが確認できます。

RTA-OS の要件は以下のものから作成されます。

- AUTOSAR SWS ドキュメント
- OSEK OS の仕様
- 顧客要件（最適化や拡張機能などに関して）
- ETAS の内部要件（ライセンス管理などに関して）

各要件に一意の識別子を割り当て、以降の処理状況を辿っていけるようにすることが重要です。AUTOSAR SWS ドキュメントにはすでにこの番号が付記されています [6]。要件管理ツールを使用すると、顧客要件と ETAS 要件の両方についてこれを徹底することができます。OSEK OS 仕様（参考文書**[6]**）の要件には番号が付いていません。そこで、ETAS は各 OSEK 要件に明示的に番号を付けたバージョンを作成しました。

RTA-OS の要件は、すべて『Unified Kernel Specification（UKS）』という文書にまとめられています。この文書により、1 つの範囲内の数値（番号）をすべての RTA-OS 要件に割り当てることができ、各要件の番号が以降のプロセスを辿るための目印になります。この仕組みを図 6-1 で説明します。

[6] AUTOSARでは多くの場合、要件番号を再利用して、異なるバージョンのSWSドキュメント内の異なる挙動を記述しています。このように、相違を一意に識別できるように統一番号方式を使用することをお勧めします。

RTA-OS ツールやターゲットポートの新しい製品バージョンごとに、コード生成ツールの挙動と生成されたコードの機能の両方について、実装や試験を通じて各要件どおりであるということを示すエビデンスが存在します。ユーザーは、RTA-OS で生成する OS コードが文書の記述どおりの挙動であることをこれらのエビデンスによって確認できるので、エビデンスは重要です。これにより、ユーザーが OS 機能全体の試験を実施する必要はなくなるので、ECU のセーフティケースを容易に構築することができます。その代わりにユーザーは、ユーザーアプリケーションが依存する RTA-OS コードの挙動（タスク／ISR のスケジューリングやハードウェアとのやりとりなど）が予想どおりに機能することを試験し、関連する要件追跡や試験の結果を ETAS から入手してセーフティケースを補強する必要があります。

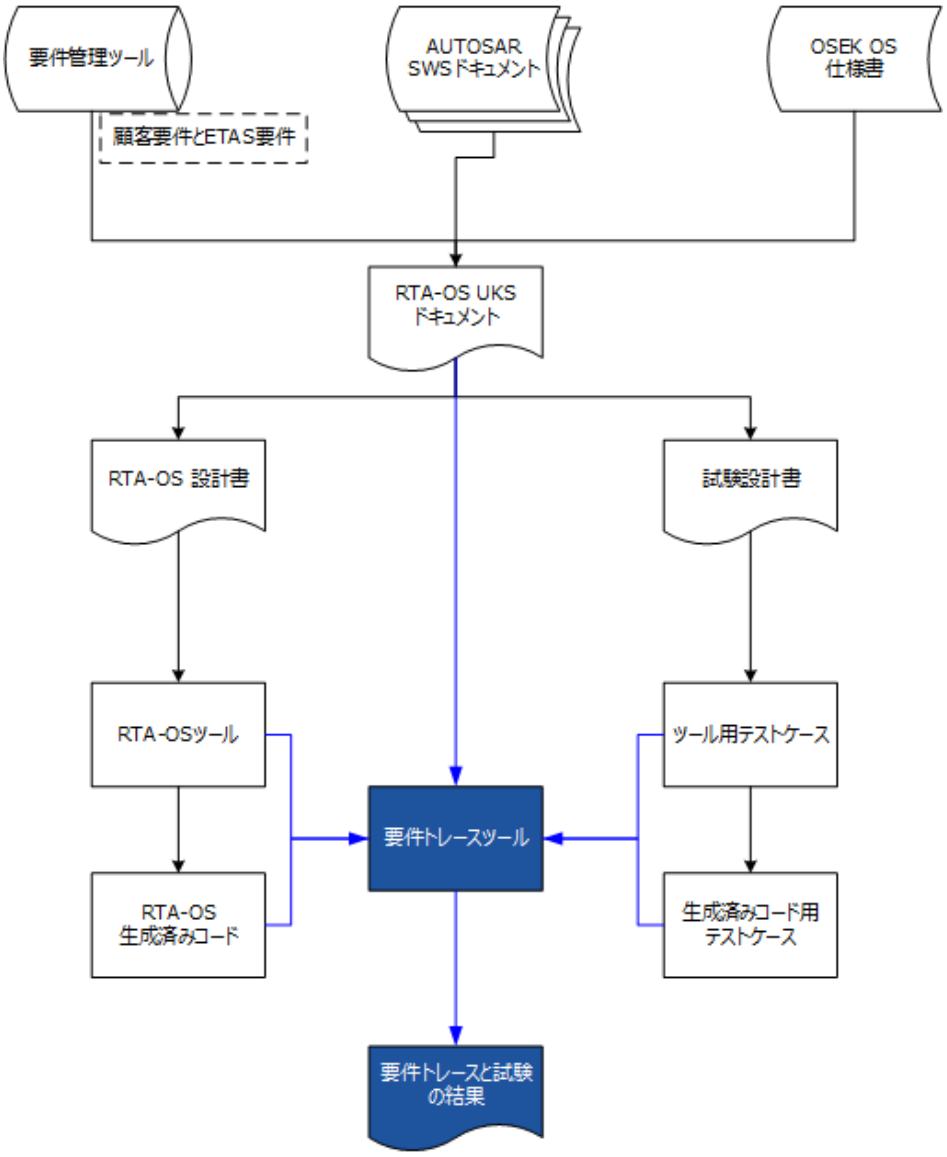

図 6-1: RTA-OS の開発と試験における要件追跡

6.3 安全性に関与する RTA-OS ツール固有の機能

ECU 開発プロセスに RTA-OS コード生成ツールを組み込むと、ECU のセーフティケースを構築する際に、RTA-OS 製品の以下のような機能を利用することができます。

複雑なハンドコーディングを自動化

RTA-OS ツールを使用して OS ソースコードを生成すると、従来は手作業で行っていたコーディングの工程を自動化することができます。RTA-OS は入力された言語に基づいて系統的

に OS コードを生成するので、入力された XML に表現されている意図と生成されたコードとの間に意味上の大きな違いは生じません。そのため、ある試験対象オブジェクトがすべてのケースで同様の挙動を取ることを 1 つの試験で十分に立証でき、変換の正当性を確認することができます。

たとえば、生成された OS に複数のタスクが含まれている場合、高優先度のタスクが低優先度のタスクをプリエンプトする際には常に同じメカニズムが使用されるので、各タスクがそれ自身より優先度の低いすべてのタスクをプリエンプトできることを明示的に試験する必要はありません。

警告！

ユーザーのシステムについて、ユーザーが選択した ISO26262 ASIL の要件に基づいて、適切なレベルの試験レビューや試験カバレッジが実現されるようにする必要があります。

オブジェクトのチェックと追跡

RTA-OS は、共通の処理を実装したルーチンのライブラリを生成します。このライブラリのソースコードには AUTOSAR SWS 要件がタグ付けされているので、第三者でも AUTOSAR SWS OS まで遡って要件を追跡することができます。これにより、たとえば、入力された XML では定義されないはずの OS 機能（ローカルサブセットや使用に関する推奨事項により禁止されている機能など）が外部入力チェックをくぐり抜けて、生成された OS コードに入り込んでしまっていないかを照合確認することができます。

ライブラリの自動コンパイル

デフォルトでは、RTA-OS は生成された OS コードをライブラリファイルにまとめます。これにより、ECU システムに統合された後の OS コードが問題を引き起こす可能性を最小限に抑えることができます。

OS をビルドする際には、ETAS が試験したものと同じコンパイラオプションが使用されるように、RTA-OS のライブラリ自動コンパイル機能を使用することを強くお勧めします。

7 ISO26262 に対するツールの評価

本章では、ISO26262 に対するツール信頼度、およびツールと RTA-OS との関係について説明します。

ISO26262 のツール信頼プロセスを図 7-1 に示します。このプロセスは以下の 2 ステップで構成されます。

1. 分析によるツールの評価
2. ツール認定

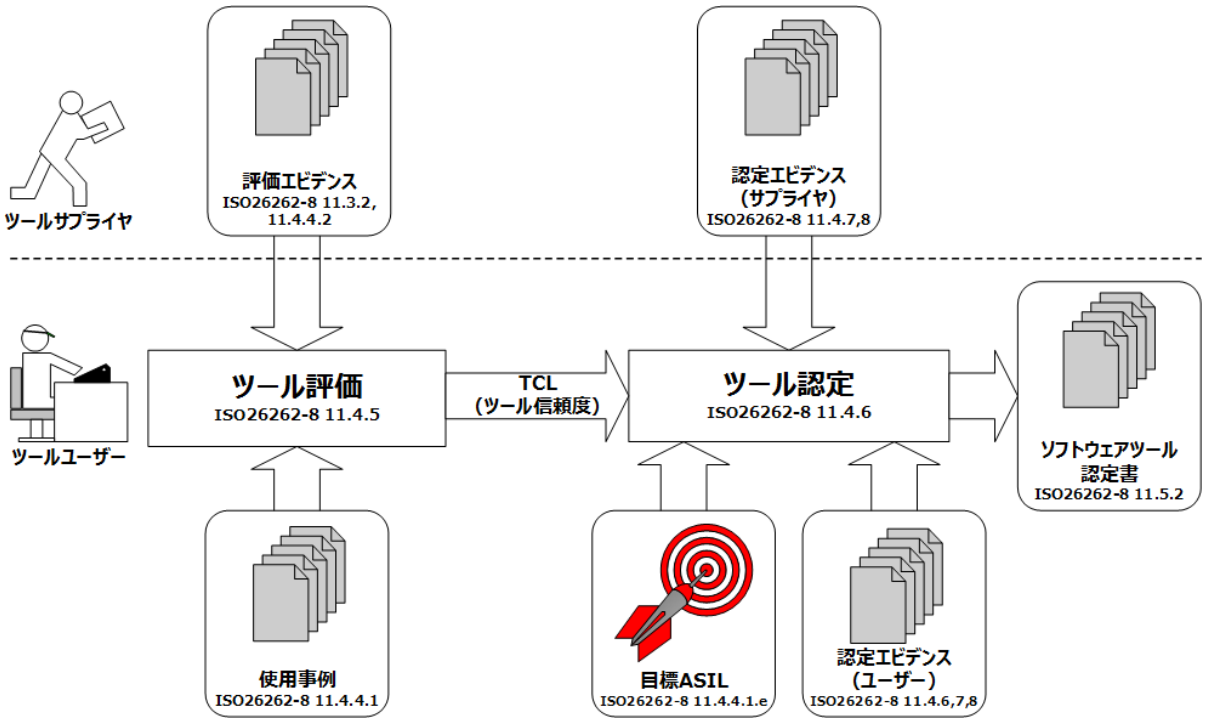

図 7-1: ツール信頼プロセス

各ステップの要件は以下の 3 つに分類されます。

1. ツールベンダーから提供される情報がないと実現できないもの。たとえば「ツールのバージョン番号を記録する」という要件を満足するには、ベンダーがツールにバージョン番号を付ける必要があります。
2. ツールユーザーによってしか実現できないもの。開発プロセスにおけるツールの使用法やインストール位置など。
3. ツールベンダーまたはツールユーザーのいずれか、または両者によって実現できる要件。

たとえば、ツール認定のためにツール評価（ISO26262-8/11.4.9）を行う必要がある場合、以下のように行うことができます。

- ベンダーが評価プロセスを実行してユーザーに提供します（ベンダーの試験結果をユーザーに提供するなど）。
- ユーザーがユーザー自身で評価プロセスを実行します。そのためには、ユーザーの使用事例を反映するツール評価スイートを ISO26262-8/11.4.9 の提案に従って開発する、といったことが考えられます。

一般的に、評価と認定にはユーザーとベンダーの両者によるエビデンスが必要となります。

これらのことについて、本章で以下のように説明します。

- 7.1 項では、ISO26262 の「分析によるツールの評価プロセス」の目的について説明します。
- 7.2 項では、製品版の RTA-OS において、ISO26262 の「分析によるツールの評価プロセス」の要件を満足していることを示す「エビデンス」を見つける方法について説明します。
- 7.6 項では、ISO26262 の要件を満足するための認定メソッドと ETAS から得られるエビデンスについて説明します。

7.1 分析によるツールの評価

分析によって評価を行う目的は、開発中のシステムにおいて安全要件違反のエラーをツールが引き起こしてしまう可能性があるかを判定することにあります。この判定結果は「ツール信頼度」（TCL: Tool Confidence Level）で表されます。TCL はターゲットシステムの ASIL と共に使用され、ツールのエラーを防ぐための安全策としてツール認定手段が必要であるか、必要な場合はどのような手段が必要か、といったことを明らかにする際に役立ちます。

ツールを評価するための分析は、システム開発プロセスの文脈において行われます。つまり、ツールチェーン内の他のツールや開発プロセス内の作業をいかに活用してツールのエラーが見逃されることを防ぐか、といったことを検討します。この評価モデルにおいては、エラーが含まれるツールであっても、開発中のシステムに影響を与えるエラーを防ぐための安全策が十分に整っているという議論が成り立つものであれば、使用することが容認されています。

ツールのエラーが開発プロセスに影響する確率を最小限にするには、以下の 2 通りの方法があります。

1. ツールのどの出力にエラーが含まれる可能性があるかを示すことにより、その出力を別のツールでチェックすることによりエラーを検出します。つまり、あるツールからの出力を別のツールへの入力としてチェックします。
2. ツール試験の結果から「問題含み」であることが判明したツール機能を使用することを回避します。

このどちらの方法を選択するかは、開発プロセス内でエラーが捕捉されることを実証することの難易度や、問題含みの機能を回避することによって生じる不便さ、といった要因によって決まります。

7.1.1 ツール信頼度（TCL）

開発プロセス内の各ツールには以下の 3 つの等級を示す TCL のいずれかを割り当てる必要があります。

TCL1 - ツールのエラーが防止／検出される信頼度が高い

TCL2 - ツールのエラーが防止／検出される信頼度が中程度

TCL3 - ツールのエラーが防止／検出される信頼度が低い

TCL は以下の 2 つの質問に対する答えによって決まります。

1. ソフトウェアツールが誤動作したり、不正な出力を生成したりした場合に、システムにおいて安全要件違反が発生するか
2. ツールの誤動作や不正な出力を、ツール自体または開発プロセス内の他のどこかで検出できるか

安全要件に違反する可能性は、「ツール故障の影響度」（TI: Tool Impact:）という指標で表され、以下のような 2 つの等級に分かれます。

TI1 - 違反の可能性なし

TI2 - 他のすべてのケース

ツールの誤動作や不正な出力を開発プロセスが検出できるか、という信頼度は「ツールエラー検出」（TD: Tool Error Detection）という指標で表され、不正な出力を検出する「確率」[7]に応じて以下の 3 つの等級に分かれます。

TD1 - ソフトウェアツールの誤動作や不正な出力が検出される信頼度が高い

TD2 - ソフトウェアツールの誤動作や不正な出力が検出される信頼度が中程度

TD3 - ソフトウェアツールの誤動作や不正な出力が検出される信頼度が低い

TI と TD の値の組み合わせを元に、図 7-2 に示す流れに従って TCL が決定されます。

TCL は、意図する目的を実現するために当該ツールが適している、という議論を構築するために必要なツール認定メソッドを表すものです。ツール認定エビデンスについては 7.6 項で詳しく考察します。本章の残りの部分では、分析による評価のために必要なエビデンスを RTA-OS 内のどこで取得できるかを説明します。

7.2 分析によるツールの評価の要件

「分析によるツールの評価」の要件は ISO26262-8/11.4.4 に記述されています。その中からツール自体に適用される要件を表 7-1 にまとめました。ISO26262-8/11.4.4.1a-f に見られる要件は、ユーザーの RTA-OS 使用計画の一部となります。RTA-OS について ETAS から提供される情報は、ISO26262-8/11.4.4.2a-f の要件によりカバーされています。

以降の項では、ツール要件について詳しく解説し、各要件（全体または一部）を満足するために RTA-OS から提供されるエビデンスについて説明します。

7.2.1 一意の識別番号（ISO26262-8/11.4.4.1a）

RTA-OS ツールとターゲットポートの各バージョンには x.y.z という形式の一意のバージョン番号が割り当てられています。x と y は製品リリースを表すバージョン番号（メジャーバージョンとマイナーバージョン）で、「RTA-OS 4.0」や「RTA-OS 5.5」のように用いられます。z は 0 から始まるリフレッシュ番号で、各バージョンのリフレッシュ（メンテナンスリリース）を表します。リフレッシュ番号の値は、RTA-OS バージョンのリフレッシュリリースが行われるたびに増加します。

RTA-OS のバージョン情報は、RTA-OS/bin フォルダでコマンドプロンプトウィンドウを開いてコマンド‘`rtaosgen --version`’を実行することによって確認することができます。このコマンドの出力には、ユーザーの PC にインストールされている RTA-OS ツールとすべてのターゲットポートのバージョン番号が含まれます。

[7] ISO26262は「確率（probability）」を「信頼度（degree of confidence）」として定義していますが、その意味や、その客観的な判断方法は示していません。しかも、信頼度「高（high）」、「中（medium）」、「低（low）」に相当する確率の値を定義していません。ISO26262の策定方法はこのように曖昧で不明確ですが、この規格の意図するところは明らかで、「ツールチェーン全体の文脈においてツールが安全要件に違反する可能性を客観的に判定する」ということです。

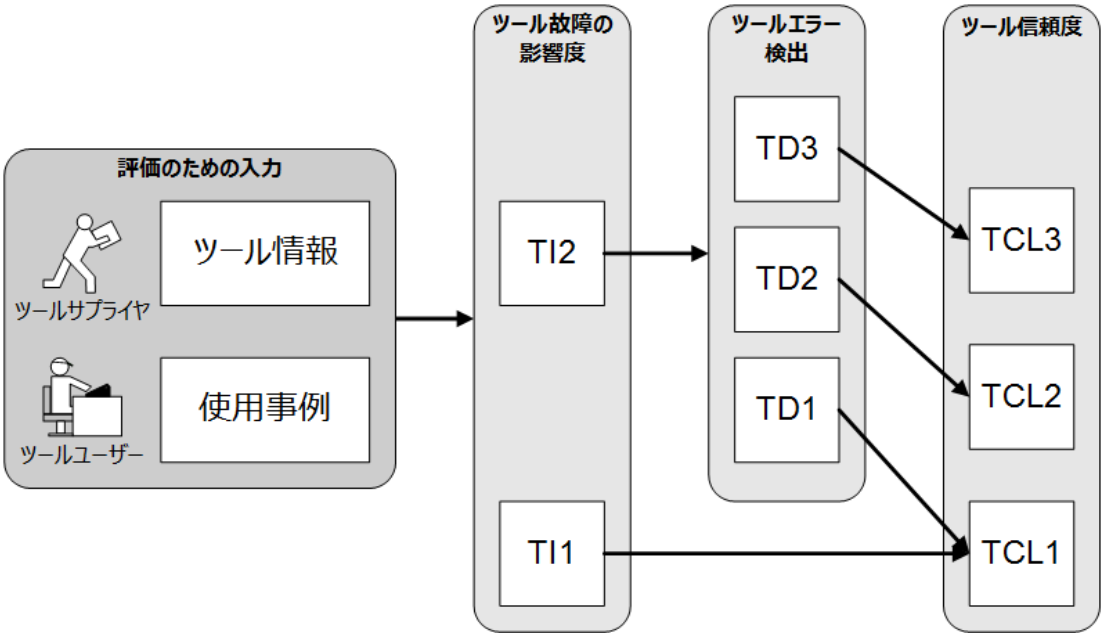

図 7-2: ツール信頼度の決定

要件	概要
ISO26262-8/11.4.4.1a	一意の識別子とバージョン番号
ISO26262-8/11.4.4.1b	コンフィギュレーション情報へのアクセス
ISO26262-8/11.4.4.1c	使用事例に関する記述
ISO26262-8/11.4.4.1d	有効な動作環境に関する記述
ISO26262-8/11.4.4.1e	違反する可能性のある安全要件の最大 ASIL に関する記述
ISO26262-8/11.4.4.1f	認定メソッドに関する記述
ISO26262-8/11.4.4.2a	機能と技術的特質に関する記述
ISO26262-8/11.4.4.2b	ユーザーマニュアル
ISO26262-8/11.4.4.2c	有効な動作環境に関する記述
ISO26262-8/11.4.4.2d	異常な動作条件下における挙動に関する記述
ISO26262-8/11.4.4.2e	既知の問題とその回避策に関する記述
ISO26262-8/11.4.4.2f	誤動作や不正出力を検出するための方策

表 7-1: ツール評価要件

7.2.2 コンフィギュレーション情報へのアクセス（ISO26262-8/11.4.4.1b）

RTA-OS は、以下に含まれるコンフィギュレーション情報に基づいて AUTOSAR OS を生成します。

1. AUTOSAR ECU パラメータ記述言語に準拠する AUTOSAR XML ファイル
2. コマンドラインオプション

RTA-OS コンフィギュレーション言語の概要は『RTA-OS リファレンスガイド（RTA-OS Reference Guide）』（参考文書**[9]**）の第 11 章、RTA-OS コマンドラインオプションについては同マニュアルの第 12 章を参照してください。

7.2.3 使用事例に関する記述（ISO26262-8/11.4.4.1c）

ISO26262 認証でカバーされている RTA-OS コード生成ツール（rtaosgen.exe）の使用事例は、以下のとおりです。

1. AUTOSAR XML ファイル（1 つまたは複数）を読み取り、有効な AUTOSAR OS が記述されていることを確認します。OS コードの生成は行いません。
2. AUTOSAR XML ファイル（1 つまたは複数）を読み取り、それに対応する AUTOSAR OS ソースコードを生成します。ソースコードのビルドは行いません。
3. AUTOSAR XML ファイル（1 つまたは複数）を読み取り、それに対応する AUTOSAR OS ソースコードを生成して、ソースコードをリンク可能なオブジェクトファイルに変換します。

RTA-OS ツール全体について考えられるその他の使用事例は、ISO26262 認証の範囲外です（2.5 項を参照）。

警告！

RTA-OS は、RTA-OS のユーザーから供給された XML コンフィギュレーションによってユーザーの要件を満たす OS が生成されることを立証するものではありません。ユーザーは ECU システム用に選択した ASIL に対応する ISO26262 要件に従って RTA-OS への入力用コンフィギュレーションの作成プロセスを管理することにより、生成される OS が確実にユーザーのシステム仕様を満たし、ユーザーの安全目標達成を支援するようにする必要があります。

7.2.4 有効な動作環境に関する記述（ISO26262-8/11.4.4.1d）

ECU システムプロジェクトで RTA-OS を使用する際には、以下の 2 通りの動作環境を考慮する必要があります。

1. RTA-OS コンフィギュレーション／コード生成ツールを格納して実行する PC の Windows 環境
2. RTA-OS で生成された OS コードを実行する ECU ハードウェア環境

RTA-OS ツールの動作環境には、ツールを格納して実行する PC とその PC 上に存在する Windows の仕様が含まれます。その環境が RTA-OS ツールに適していることを検証するには、ETAS による RTA-OS のリリース試験に使用された環境に関する記述を確認してください。これは『RTA-OS 入門ガイド（RTA-OS Getting Started）』（参考文書**[8]**）に記述されています。

RTA-OS で生成された OS コードの動作環境には以下の要素が含まれます。

- マイクロコントローラ
- マイクロコントローラハードウェアインターフェース
- 所定のバージョンのコンパイラツールチェーン
- 上記コンパイラツールチェーンの設定に使用されるコマンドラインスイッチ

警告！

ECU システム環境が RTA-OS で生成された OS コードの実行に適していることを検証するには、ETAS による RTA-OS の当該ターゲットポートのリリース試験に使用された環境に関する記述を確認してください。これは所定の『RTA-OS Port Guide』という文書に記述されています。

ETAS で使用された試験環境をどこで確認できるかは、7.2.9 項に記述されています。ユーザーの ECU システム環境が ETAS による RTA-OS の試験で使用された環境と異なる場合は、環境の相違に起因して ECU システムの動作中に安全関連の問題が発生する一切の可能性を回避するため、ユーザー環境における同等のリリース試験の実施を ETAS に委託することを強くお勧めします。

7.2.5 最大 ASIL（ISO26262-8/11.4.4.1e）

ETAS は、RTA-OS が、ASIL D の要件を満たす ECU システム内で使用される OS の生成に適していると想定しています（**想定 1** を参照）。従って、RTA-OS で生成されたコードが誤動作した場合に違反する可能性のある安全要件の最大 ASIL は、ASIL D になります。

7.2.6 認定メソッドに関する記述（ISO26262-8/11.4.4.1f）

ETAS は RTA-OS を TCL1 に分類しています（7.5 項を参照）。ISO26262-8/11.4.6.1 によれば、これ以上の認定メソッドは必要ありません。しかし、ユーザー独自の判断でそれ以外の TCL に RTA-OS を分類する場合は、7.6 項に示されている認定メソッドについての情報を参照してください。

7.2.7 機能と技術的特質に関する記述（ISO26262-8/11.4.4.2a）

RTA-OS の機能と技術的特質は以下の文書に完全に記述されています。

- 『RTA-OS 入門ガイド（RTA-OS Getting Started）』（参考文書**[8]**）
- 『RTA-OS ユーザーガイド（RTA-OS User Guide）』（参考文書**[10]**）
- 『RTA-OS リファレンスガイド（RTA-OS Reference Guide）』（参考文書**[9]**）
- 『RTA-OS Port Guide』（各 RTA-OS ターゲットポート用に 1 部）

これらの RTA-OS ユーザー文書の所在については、7.2.8 項を参照してください。

7.2.8 ユーザーマニュアル（ISO26262-8/11.4.4.2b）

RTA-OS 用の汎用ユーザーマニュアルは、図 7-3 に示すように、すべて RTA-OS\Documents フォルダに保存されています。

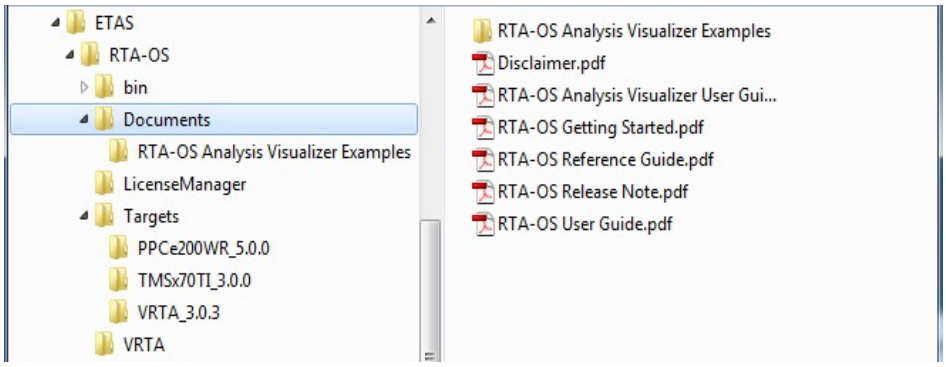

図 7-3: RTA-OS のユーザーマニュアルの保存場所

RTA-OS ターゲットポート別のユーザーマニュアルは、図 7-4 に示すように、各ターゲットポート用のフォルダに保存されています。

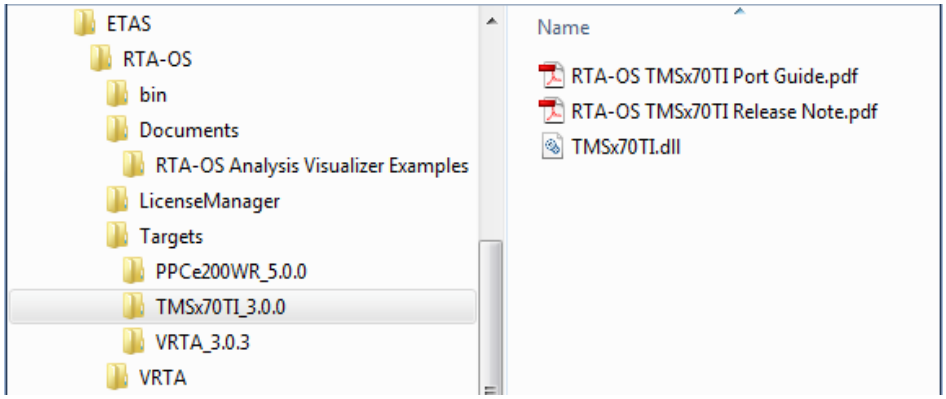

図 7-4: RTA-OS ターゲットポートのユーザーマニュアルの保存場所

7.2.9 有効な動作環境に関する記述（ISO26262-8/11.4.4.2c）

RTA-OS では、以下の 2 種類の動作環境を考慮する必要があります。

1. RTA-OS コンフィギュレーション／コード生成ツールの動作環境
2. 生成された OS の動作環境（RTA-OS ターゲットポートごとに異なります）

RTA-OS コンフィギュレーション／コード生成ツールに有効な動作環境は『RTA-OS 入門ガイド（RTA-OS Getting Started）』の第 2 章に記述されています。この環境にはツールの格納と実行に使用される PC の仕様に関するハードウェア環境と、ツールの実行に必要な Microsoft Windows インスタレーションに関するソフトウェア環境とが含まれます。

各 RTA-OS ターゲットポート用に有効な動作環境は、各ターゲット用の『RTA-OS Port Guide』という文書に記述されています（7.2.8 項を参照）。この環境には以下の要素が含まれます。「OS コード」は、RTA-OS ツールで生成された OS コードを指します。

- ターゲットマイクロコントローラ
- OS コード内で想定されているマイクロコントローラに接続するためのハードウェアインターフェース
- OS コードのコンパイルに使用されるコンパイラツールチェーンバージョン
- OS コードのコンパイルに使用されるコンパイラツールチェーンスイッチ

警告！

ユーザーが ECU システム内で使用しようとする動作環境が、ETAS による RTA-OS の試験で使用された動作環境と一致することを確認する必要があります。少しでも異なる点がある場合は、試験環境の相違に起因して ECU システムの動作中に安全関連の問題が発生する可能性をすべて回避するため、ユーザー環境における同等のリリース試験の実施を ETAS に委託することを強くお勧めします。

7.2.10 異常な動作条件下における挙動に関する記述（ISO26262-8/11.4.4.2d）

RTA-OS ツールの内部に矛盾が存在すると、深刻な障害が発生します。そのような際には `Diagnostic.dmp` という名前の診断ログファイルが生成されるので、ETAS が故障の原因を特定する際にこのファイルが役立ちます。

RTA-OS コンフィギュレーションのエラー（モデルに使用されている AUTOSAR モデリング言語の構文やセマンティクスが不正であったり、互いに両立しないオプションが共に選択されていたりする状態）はコード生成／ビルド時にツール自体によってチェックされ、RTA-OS コンフィギュレーションツールのエラービューアまたはコマンドラインに報告されます。

情報不足によりコンフィギュレーションが予想どおりの挙動を取らない可能性があることを RTA-OS が検知した場合は、RTA-OS コンフィギュレーションツールのエラービューアまたはコマンドラインに警告メッセージが出力されます。

RTA-OS ツールについて考えられる故障モードについての詳細情報は 6.1 項に記述されています。

7.2.11 既知の問題とその回避策に関する記述（ISO26262-8/11.4.4.2e）

バグ追跡

ETAS 製品に関する問題は、内部で報告されたものもユーザーにより報告されたものも、すべて「ETAS Helpdesk International」（EHI）という ETAS のバグ追跡システム内で管理されます。報告された問題には EHI<番号>という形式の一意の番号が割り当てられます。

リリースノート

RTA-OS 製品の各リリースにはリリースノートが添付されます。この文書には以下のような事項が記述されています。

- 新機能
- 当該リリースに関する既知の問題
- 当該リリースにより修正された問題（参照用に EHI 番号が付記されています）

RTA-OS のコンフィギュレーション／コード生成ツールのリリースノート は、図 7-3 に示すようにインスタレーションの Documents フォルダに保存されています。各 RTA-OS ターゲットポートのリリースノートは、図 7-4 の例のようにインスタレーションの当該ターゲットのフォルダに保存されています。

既知の問題に関するレポート（KIR）

ETAS はリリース後の RTA-OS に関して明らかになった重大案件（量産システムのリコールにつながる可能性があるような問題など）は、一般の問題とは区別して扱われ、既知の問題に関するレポート（KIR: Known Issue Report）に登録されます。

KIR には、発生した重大問題ごとに以下のような内容が記述されます。

- EHI 番号
- 問題に関する詳細（該当バージョンやモジュール、具体的な症状、発生条件など）
- 問題の回避策や次善策（それらが存在する場合）
- 問題の修正予定

KIR に新しい案件が追加された際と既存の案件が解決された際には、ユーザーに対して E メールでその旨が通知されます。KIR は http://www.etas.com/kir からダウンロードすることができます。図 7-5 に示すように、**製品ファミリ / カテゴリ** 欄で **RTA**、**製品名 / トピック** 欄で **RTA-OS** を選択して **Go** をクリックすると、RTA-OS に関する KIR の一覧が表示されます。

ダウンロードパスワードは、製品に添付されたドキュメントをご参照ください。

図 7-5: ETAS ダウンロードセンターから KIR をダウンロード

ホットフィックス

「ホットフィックス」は RTA-OS 用に正式にリリースされたパッチです。

新しいホットフィックスは E メールでユーザーに通知されます。すべてのホットフィックスは ETAS ダウンロードセンター http://www.etas.com/ja/products/download_center.php からダウンロードすることができます。RTA-OS のホットフィックスの一覧を表示するには、**製品ファミリ / カテゴリ** 欄で **RTA**、**製品名 / トピック** 欄で **RTA-OS** 、**カテゴリ** 欄で **ホットフィックス / サービスパック** を選択して **Go** をクリックします。

警告！

RTA-OS の各ホットフィックスは、ユーザープロジェクトの状況と問題の重要度を照らし合わせたうえで、適切であると判断された場合にのみ使用するようにしてください。不適切なホットフィックスをインストールすると、それによる悪影響が生じる危険性もあります。

RTA-OS のリリースノート（上記参照）には、修正されたバグと前回のリリース以降の主な変更について明記されています。

7.2.12 誤動作や不正な出力を検出するための方策（ISO26262-8/11.4.4.2f）

図 3-3 に示すように、RTA-OS は XML コンフィギュレーションファイルに従って、AUTOSAR OS の C コードとアセンブラコードを生成します。ツールの誤動作の原因となるコンフィギュレーション内の矛盾は、7.2.10 項の記述に従って処理されます。

コード生成エラーを最小限に抑えるため、RTA-OS は詳細にわたる試験が実施された後にリリースされますが、ETAS は RTA-OS でコード生成エラーが発生しないことを保証することはできません。

コード生成エラーは以下の 2 種類に分かれます。

1. 構文違反のある C コードやアセンブラコード
2. 構文的には有効でもセマンティクス（OS 挙動）が不正なコード

構文違反のある C コードとアセンブラコードはコンパイラツールチェーンにより検出されつため、これらのエラーが ECU システムに入り込むことはありません。

誤った OS 挙動を招くコードは、ETAS でのリリース試験においてある程度検知できます。ETAS は RTA-OS のリリース試験において、生成される OS が常に AUTOSAR OS 仕様に従って挙動するように確認することを試みます。しかし実際に使用されるコンフィギュレーションには非常に多くのパターンがあるため、リリース試験では、すべてのコンフィギュレーションパターンをいくつかの同値クラスに分類し、各クラスを代表する一連のテストケースを定義することに重点を置いています。

たとえば、システム内で定義されているタスクの数が 32 個までの場合は、RTA-OS はタスクを表現するために 32 ビット変数を生成し、タスクの数が 33 個以上になると、代わりに 64 ビット変数を生成します。そこで重要となるのはタスク数が 32 のテストケースと 33 のテストケースで、これらは「コーナーケース」と呼ばれます。コーナーケースが正しく機能することがわかれば、他のすべてのケース、つまり、タスク数が 1 から 31 までと 34 から 64 までのケースの挙動は正しくなると予想できます。

警告！

ユーザーの ECU システム開発プロセスにおいて、発生する可能性のあるすべての RTA-OS コード生成エラーをすべて検知できるようにすることが非常に重要です。ワークフローの中でそのようなエラーが見逃されてしまう可能性を最小限に抑える RTA-OS の使用方法は、第 4 章で説明されています。

さらに、ユーザーが選択した ASIL に適したレベルのレビュー、試験およびカバレッジが開発プロセスに含まれていることを確認する必要があります。

7.3 RTA-OS のツールインパクト（TI）

ISO26262-8/11 に基づき、ETAS は RTA-OS を TI2 に分類しています。これは、RTA-OS のコード生成機能のバグが原因でアプリケーションソフトウェアのエラーが引き起こされる可能性があるためです。さらに、RTA-OS ツールのバグにより入力エラーが検出できなかった場合は、エラーメッセージが出力されずにそのまま OS が生成されてしまう可能性があります。

7.4 RTA-OS のツールエラー検出（TD）

ISO26262-8/11 に基づき、ETAS は RTA-OS を TD1 に分類しています。これは、RTA-OS がコードジェネレータであるため、その出力を ASIL D 開発に要求される手法に従ってシステムの文脈で検証する必要があり、これによって十分に潜在的エラーを検知することができるためです。

ツールエラーには以下のようなタイプがあります。

エラー	検出方法
RTA-OS の不正な使い方（コマンドラインオプションの間違いや矛盾など）	RTA-OS が内部的に検出し、ユーザーに FEWI メッセージで報告します。
不正または矛盾のある XML 入力	RTA-OS が内部的に検出し、ユーザーに FEWI メッセージで報告します。
不正な OS 生成	RTA-OS 単体試験
不正な OS 生成と単体試験	ISO26262-6/10、12 に基づいてユーザーが行う結合試験とソフトウェア安全要件評価

表 7-2: RTA-OS コード生成時に考慮されるエラー

ETAS は、ユーザーの ECU システム開発プロセスを支援する目的で、RTA-OS の試験レポートを提供します。エンジニアリングサービス契約が結ばれている場合は、ユーザーの ECU システムと同じ条件で RTA-OS の試験を繰り返し行い、必要に応じて RTA-OS のテストケースをご利用いただくことも可能です。

警告！

ETAS はユーザーの開発プロセスの構造を知り得ないため、ユーザーは、OS コード生成ツールとしての RTA-OS のツールエラー検出（TD）レベルを自主的に判定する必要があります。

ETAS はユーザーがシステムの中で AUTOSAR OS のどの機能を使用するかを知り得ないため、ユーザーは、生成された OS コードについて、システム内の他のコードと同様に、ISO26262 に定められたコードのレビューや試験を実施する必要があります。

7.5 RTA-OS のツール信頼度（TCL）

7.3 項と 7.4 項の結果、およびこれまでに述べられている「生成されたコードは、ユーザーが選択した ASIL に対応する ISO26262 要件に従ってユーザーの開発プロセスの中で試験されレビューされているものとする」という想定に基づき、ETAS は表 7-3 に示すように、RTA-OS のツール信頼度を TCL1 に分類しています。

		ツールエラー検出		
		TD1	TD2	TD3
ツールインパクト	TI1	TCL1	TCL1	TCL1
	TI2	TCL1	TCL2	TCL3

表 7-3: RTA-OS のツール信頼度の決定

RTA-OS の TCL に関するユーザーの判定結果がこれと一致する場合は、ISO26262-8/11.4.6.1 に基づき、これ以上の認定メソッドは必要ありません。

一方、RTA-OS の TCL が 2 または 3 であるとユーザーが判断した場合は、ISO26262-8/11.4.6.2 に記述されているツール認定手段を実施する必要があります。以降の項では、ETAS がその認定を支援する方法について説明します。

7.6 ツール認定

ツール分類で割り当てられた TCL と開発中のシステムの ASIL を基に、ツールに適した認定メソッドを示します。「認定」とは、システムのセーフティケースにおいて、ツールが安全に使用できるものであるという議論（argument）を明確にするための手段です。

ISO26262 は以下の 4 つの認定メソッドを定めています。

1. 使用実績に基づく信頼性向上
2. 開発プロセスの評価
3. ソフトウェアツールの評価
4. 安全規格に従った開発

ISO26262 は、それぞれの TCL／ASIL の組み合わせにどのメソッドが適しているか、という指針を示しています。その概要を、以下の記号を使用して表 7-4 と表 7-5 に示します。

- \- 推奨しません
- \+ 推奨します
- ++ 強く推奨します

ここでは以下の事項に注目してください。

- TCL1 に分類されているツールについては更なる認定手段が必要ないこと
- より上位の TCL や ASIL に分類されるために必要な認定エビデンスのレベル
- 各認定メソッドを選択可能であること。ユーザーの開発プロセスで使用するツールの認定に適したメソッド（またはメソッドの組み合わせ）を選択することができます。ただしユーザーは、ユーザーのセーフティケースにおいて、選択したメソッドが、ツールが安全要件に違反する状況に陥ることがないことを立証するのに適している、という議論を提供する必要があります。

以降の項では、ツール認定エビデンスに関して ETAS が支援できる内容を説明します。

メソッド	ASIL A	ASIL B	ASIL C	ASIL D
使用実績に基づく信頼性向上	++	++	++	+
ツール開発プロセスの評価	++	++	++	+
ソフトウェアツールの評価	+	+	+	++
安全規格に従った開発	+	+	+	++

表 7-4: TCL2 のための認定メソッド

メソッド	ASIL A	ASIL B	ASIL C	ASIL D
使用実績に基づく信頼性向上	++	++	+	+
ツール開発プロセスの評価	++	++	+	+
ソフトウェアツールの評価	+	+	++	++
安全規格に従った開発	+	+	++	++

表 7-5: TCL3 のための認定メソッド

7.6.1 使用実績に基づく信頼性向上

使用実績に基づく信頼性の向上については、このツールのある特定のバージョンが、類似の（同一であることが好ましい）ツールチェーンと開発プロセスにおいて、同じタイプのシス

テム向けにすでに使用されていること、さらにそのツールがこれまでにどの安全要件にも違反していないことを示すエビデンスが必要です。

使用実績に基づき信頼性を証明する議論は、ツールの使用形態やツールが使用されるプロジェクトに関する詳細情報に依存します。必要なエビデンスを収集し、使用実績に基づき信頼性を証明する議論を構築するのはユーザーの責務であり、ETAS はそれを支援するための情報（またはその一部）を提供することができます。

ISO26262 は、ツールの履歴に関するエビデンスと、ツールに関する情報の分析と評価を要求しています。求めてられいるエビデンスの要件は表 7-6 にまとめられています。表 7-7 には、RTA-OS の過去の使用履歴を考慮して分析／評価する際の ISO26262 の要件がまとめられています。どちらの表にも ETAS が提供できる支援内容が示されています。

要件	概要	ETAS からの支援内容
11.4.7.2a	類似した目的のために、同等の使用事例において、同等の環境および類似した機能制約条件の下で使用されている	RTA-OS が類似したプロジェクトと環境で使用されているか、という情報を NDA に基づいて提供できます。
11.4.7.2b	十分かつ適切なデータがある	ある特定の RTA-OS バージョンのユーザー数、およびその中で類似の目的で RTA-OS を使用しているユーザーの数に関する情報を、NDA に基づいて提供できます。
11.4.7.2c	ツール仕様が変更されていない	RTA-OS の仕様変更については、リリースノートに詳述されています（7.2.11 項を参照）。
11.4.7.2d	誤動作や不正な出力の文書化	RTA-OS の誤動作や不正な出力について、KIR（既知の問題に関するレポート）やリリースノートに情報を記録しています（7.2.11 項を参照）。

表 7-6: 使用実績に基づく信頼のエビデンス要件

要件	概要	ETAS からの支援内容
11.4.7.3a	一意の識別子と一意のバージョン番号	RTA-OS のバージョン情報については 7.2.1 項に記述されています。
11.4.7.3b	これまでのプロジェクトで使用されたコンフィギュレーションの情報	該当なし。ユーザーによる過去の RTA-OS の使用事例についての詳細を提供することはできません。
11.4.7.3c	使用期間と使用に関するデータ	所定バージョンの RTA-OS（またはそのターゲットポート）が市販されていた期間の情報は、ETAS から提供できます。製品の使用権が認められているユーザーの数は NDA に基づいて提供できますが、使用事例はそれぞれ異なっている可能性があるので、このエビデンスがそのままユーザーの議論を支持できるとは限りません。
11.4.7.3d	誤動作や不正な出力についてまとめた文書	既知の問題に関するレポート（KIR）およびリリースノートに、ETAS で把握している問題の一覧が記載されます。詳細は 7.2.11 項を参照してください。
11.4.7.3e	過去のバージョンで修正された誤動作の一覧	所定バージョンの RTA-OS で解決された誤動作の一覧は各バージョンのリリースノートに記載されています。詳細は 7.2.11 項を参照してください。
11.4.7.3f	不正な出力の回避策や次善策	既知の問題に関するレポート（KIR）には、問題の回避策や次善策がある場合はそれらが明記されます。詳細は 7.2.11 項を参照してください。

表 7-7: 使用実績に基づく信頼の情報要件

7.6.2 開発プロセスの評価

開発プロセスの評価では、当該ツールが国家規格や国際規格に対応して開発されていることと、規格への適合性がアセスメントにより実証されていることが要求されます。これに該当する 2 つの要件を表 7-8 に示します。

ISO26262 は、どのプロセス規格を使用するか、また複数の適合レベルが設けられている規格（CMMI L1-5 など）においてどの適合レベルが必要であるかを定めていません。

警告！

優れたプロセスが必ずしも製品の安全性を保証するとは限りません。ユーザーのツール認定の議論においては、その製品がどのようにして開発されたかということだけではなく、そのツールがユーザーの使用事例に関して正しく機能するかどうかを検討する手段も含めることをお勧めします。

要件	概要
11.4.8.2	開発プロセスは適切な規格に準拠しているべきである
11.4.8.3	開発プロセスの評価は、適切な国家規格や国際規格に基づくアセスメントにより行われるべきである

表 7-8: 開発プロセス評価の要件

RTA-OS を含むすべての ETAS 製品は、ETAS 製品エンジニアリングプロセス（PEP）に従って開発されています。表 7-9 は ETAS PEP の各ステージとステージ分けされた CMMI Dev v1.2 の各ステージとの対応関係を示しています。

アセスメントによる評価に関する要件（ISO26262-8/11.4.8.3）を満たすには、以下の 2 通りの方法があります。

1. ユーザーがツールメーカーを監査する
2. ユーザーがツールメーカーに、第三者監査のエビデンスを提供するよう依頼する

これらの可能性について、以降の項で考察します。

PEP ステージ	CMMI プロセスの範囲
製品管理	PP (Project Planning)
プロジェクト管理	PP (Project Planning) PMC (Project Control and Monitoring) MA (Measurement and Analysis)
サプライヤ管理	SAM (Supplier Agreement Management)
要件エンジニアリング	REQM (Requirements Management) RD (Requirements Development)
製品開発	TS (Technical Solution) PI (Product Integration) VER (Verification)
製品の試験とリリース	VER (Verification) VAL (Validation)
コンフィギュレーションの管理とレビュー	CM (Configuration Management) VER (Verification)
上記以外の手順	PPQA (Process and Product Quality Assurance) MA (Measurement and Analysis) OPF (Organizational Process Focus) OPD (Organizational Process Definition) OT (Organizational Training) DAR (Decision Analysis and Resolution)

表 7-9: 開発プロセスの評価の要件

ツールメーカーを監査する

ユーザーが選択した品質プロセスに照らしてツールベンダーを監査することにより、ツールベンダーの内部プロセスを最大限に可視化できます。ETAS はプロセス監査に対応できるので、これまでにもユーザーの QA 部門が AutomotiveSPiCE、CMMI などに基づいて行うアセスメントを受けています。監査の実施については ETAS にご相談ください。

独立した第三者による監査

ユーザーが関わるすべてのツールベンダーを監査するには、非常に多くの費用と時間がかかります。この工数は、ツールベンダーの監査が独立した第三者機関によって行われ無事終了したことを示すエビデンスをツールベンダーが提供することによって省略できます。

第三者監査のエビデンスとして提供を受けることができるものの内容については、ETAS にお問い合わせください。

7.6.3 ソフトウェアツールの評価

ソフトウェアツールの評価においては、そのツールがどのようにして構築されたかということから、そのツールが使用目的に適しているかということに焦点が移ります。要件は ISO26262-8/11.4.9 に記述されています。

どのような評価も、製品が使用目的に適しているかどうかを明確にすることを目的としています。つまり「これは的確な製品であるか？」という質問に答えることになります。

ユーザーは「システムインテグレータ」であり、RTA-OS を評価する際には、RTA-OS が使用に適していることを保証するためのすべての評価作業に対する責任があります。ユーザーは、7.2.3 項に記載されている認証済み使用事例の一覧を参照して、ツールに期待する機能を定義し、検査、試験、実行可能性の検討などを行って、RTA-OS がユーザーの要求どおりに機能することを立証する必要があります。ユーザーによる評価の結果は一般的に、ユーザーのツールチェーンのセーフティケースの一部として使用できるものとなります。

ツールがユーザー要件に適したものであるように見える場合は、評価を行うことにより、所定のケースについてツールが的確に機能するかどうかを明確にすることができます。

表 7-10 は、ソフトウェアツールの評価に関する ISO26262 の要件の概要と、それに関連して ETAS で行われる RTA-OS の検証と妥当性確認の内容を示しています。

RTA-OS のテストについてのより詳細な情報は、ご要望に応じて、所定の契約と NDA に基づきご提供させていただきます。

要件	概要	ETAS からの支援内容
11.4.9.2a	ツールが要件に準拠していることの評価	各バージョンの RTA-OS は、自動化された詳細な試験の実施後にリリースされます。試験は、検証（製品が要件を満たしていることを確認すること）と妥当性確認（コード生成プロセスの完全性を確認すること）の両面をカバーしています。
11.4.9.2b	評価中に明らかになったエラーの分析	試験中に明らかになったエラーは、試験手順の一環として調査され、記録されています。外部から確認できる結果（不正なコードなど）をもたらすエラーは、すべてリリースノートに記録され（7.2.11 項を参照）、リリース後に確認された重大問題は「既知の問題に関するレポート（KIR）」によりすべてのユーザーに通知されます（7.2.11 項を参照）。
11.4.9.2c	異常な動作条件に対する反応	RTA-OS の試験では、異常な入力が正しく検出されて報告されることが確認されています。

表 7-10: ソフトウェアツールの要件に対する妥当性確認とエビデンス

7.6.4 安全規格に従った開発

RTA-OS の開発環境は、安全規格に従っていません。

第 3 部 RTA-OS で生成したコードの使用に関する考察

8 組み込みソフトウェアの文脈における RTA-OS

以降の項では、RTA-OS をユーザーのソフトウェア開発プロセスに統合する方法について説明します。

8.1.1 RTA-OS によるコード生成

RTA-OS は AUTOSAR OS 用の完全な C ソースコードを生成します。このソースコード生成プロセスは以下の 2 つの部分で構成されます。

1. **コアコード生成:** コアコードはターゲットのマイクロコントローラとコンパイラに依存しないコードで、OS ソースコードの約 95%を占めます。
2. **ターゲットコード生成:** ターゲットコードは特定のマイクロコントローラファミリと特定のコンパイラツールチェーンに対応するためのコードで、OS ソースコードの約 5%を占めます。

インストールされた RTA-OS は、コアコード生成を実装するコアツール一式と、1 つまたは複数のターゲットポートとで構成されます。1 つのターゲットポートには、特定のマイクロコントローラファミリ（同じ CPU コア、割り込みコントローラ、命令セットアーキテクチャを持つマイクロコントローラ群）と特定のサードパーティコンパイラツールチェーンに対応するターゲットコード生成機能が含まれます。各ターゲットポートは、十分な試験が行われた後に、RTA-OS コアコード生成ツール用アドオンとして提供されます。

ETAS は、ターゲットポートごとに生成された RTA-OS コードについて、所定のマイクロコントローラバリアント上で、所定のコンパイラバージョンとコンパイラスイッチを用いて試験を行います。試験されたマイクロコントローラバリアント、コンパイラバージョン、コンパイラスイッチの詳細については、各ターゲット用の『RTA-OS Port Guide』 に記述されています（ドキュメントの保存場所は 7.2.8 項を参照してください）。典型的な ECU プロジェクトの構築においては、RTA-OS は、OS ソースコードを生成する際と、その後にサードパーティコンパイラツールチェーンを起動して OS ソースコードをコンパイルしてライブラリにリンクする際に使用されます。その際には、リリース試験用に使用されたのと同じコンパイラスイッチが ECU システムコード用にも使用されることが保証されています。

OS ソースコードの生成のみを行う RTA-OS オプション機能を使用し、コンパイルとリンクは RTA-OS 外部で行う場合は、必ず同じコンパイラバージョンとスイッチを使用する必要があります。

RTA-OS により生成されたコードには以下のような特徴があり、「提供される入力に対して正しいコード生成プロセスが行われる」というセーフティケースのエビデンス収集に役立ちます。

モジュール構造

RTA-OS により生成されるソースコードのアーキテクチャ構造は AUTOSAR OS の API ディスクリプションと一致し、各 API 関数はそのソースファイルの中に含まれています。そのため、生成されたコード内において、コンフィギュレーション内の各オブジェクトに該当する部分を容易に特定することができます。

MISRA-C:2004準拠コード

RTA-OS は MISRA-C:2004 [5]に対する逸脱レポート（deviation report）を生成できます。これは、設定された OS に関する逸脱項目の一覧をまとめたもので、どのルールからどのように逸脱しているか、逸脱が発生した理由と回数、またどのソースファイルで逸脱が発生しているか、といった情報が含まれています。

RTA-OS などのツールによって生成されたコードを含め、ECU システムのすべてのソースコードは、サードパーティの MISRA 準拠チェッカを使用して状態を調べることを強くお勧めします。

RTA-OS は、設定された OS について算出されたワーストケースのスタック使用量を報告するレポートを生成できるので、ワーストケースのスタック使用量に十分なメモリがリンカによりアロケートされていることを確認できます。RTA-OS のスタック監視機能により、生成された OS コードが所定の限度を超えないかを確認することが可能です。

スタックの情報は、ユーザーのコンパイラツールチェーンや他のランタイム監視ツールからも得ることができます。複数のソースから取得した情報を比較し、ECU システムに必要なスタックのメモリ量を正しく把握することをお勧めします。

9 RTA-OS により生成されたコードに関する想定集

本章では、安全関連システムの開発時に RTA-OS ツールにより生成された OS ソースコードの使用に関して、ETAS が想定する条件を紹介します。これらの想定のうちのいずれかがユーザーの ECU システムに該当していないと思われる場合は、ユーザーシステムに変更を加えるか、ETAS に連絡して（連絡先は第 11 章を参照）その想定がなされた根拠について確認してください。

想定事項の多くには、その想定を有効にするためにユーザーが履行すべき「**責務**」が併記され、さらにユーザーが責務を履行する際に役立つ「**方策**」が提示されているものがあります。

9.1 RTA-OS により生成されたソースコードの使用に関する想定

コンフィギュレーション

想定 31: RTA-OS により生成されたコードとやりとりするすべての BSW とアプリケーションソフトウェアは、同じ標準 AUTOSAR ヘッダファイルを使用するものとします。

責務 31: ECU システムを構成するすべてのソフトウェアに、同じ標準 AUTOSAR ヘッダファイル、つまり以下のファイルをインクルードする必要があります。

- Compiler.h
- Compiler_Cfg.h
- PlatformTypes.h
- MemMap.h

方策 31a: AUTOSAR 標準ヘッダファイルの使用と内容をレビューして、ECU システムを構成するすべてのソフトウェアに同じファイルがインクルードされていることを確認してください。

ハードウェアインターフェース

想定 32: マイクロコントローラレジスタは、RTA-OS により生成されたコードが起動される前に、各ターゲット用の『RTA-OS Port Guide』の記述に従って初期化されているものとします。

責務 32: RTA-OS の `StartOS()` API 関数が呼び出される前に、マイクロコントローラレジスタが各ターゲット用の『RTA-OS Port Guide』の記述に従って正しく初期化されていることを確認する必要があります。

方策 32a: 各ターゲット用の『RTA-OS Port Guide』を参照してシステムコードをレビューしてください。必要に応じて ETAS に相談してください。

想定 33: RTA-OS により生成されたコードが起動された後、各ターゲット用の『RTA-OS Port Guide』に記述されているマイクロコントローラレジスタがシステムによって変更されることはないものとします。

責務 33: 各ターゲット用の『RTA-OS Port Guide』に記述されているマイクロコントローラレジスタが、ランタイムに RTA-OS 以外のどのシステムソフトウェアによっても変更されないようにする必要があります。

方策 33a: 各ターゲット用の『RTA-OS Port Guide』を参照してシステムコードをレビューしてください。ご不明の点は ETAS にご相談ください。

想定 34: 生成された RTA-OS とターゲットハードウェアとの間のインターフェースは、コールバック関数として実装します（参考文書**[9]**『RTA-OS リファレンスガイド

（RTA-OS Reference Guide）』）。これらのコールバック関数の内容は、すべてアプリケーションにとって正しいものであるものとします。

責務 34:　生成された OS からユーザーコードへのすべてのコールバックをレビューして試験し、要求されている機能が実行されることを確認してください。要求されている機能は、RTA-OS ユーザードキュメントや RTA-OS により生成される情報メッセージに明示されています。

方策 34a:　要求されているコールバックコードをレビューして、必要な機能とハードウェアとのやりとりが実装されていることを確認してください。

スケジューリング

想定 35:　適切なハードウェアタイマがマイクロコントローラ上に存在し、RTA-OS により生成されたコードに対して RTA-OS ツールで設定されたすべてのカウンタ／アラームとスケジュールテーブルの挙動を制御するのに適した割り込みを生成するようにシステムによって設定されるものとします。どのタイマを使用するかは、システムインテグレータが決定します。

責務 35:　適切なハードウェアタイマがマイクロコントローラ上に存在していること、およびシステムに設定されているカウンタ／アラームおよびスケジュールテーブルを駆動するために適切な‘tick’または‘advance’割り込みが設定されていることを確認する必要があります。

方策 35a:　マイクロコントローラのリファレンスマニュアルを参照してスケジュール要件をレビューし、‘tick’割り込みや‘advance’割り込みに必要なレートと分解能が、選択されているタイマに提供されていることを確認してください。

方策 35b:　カウンタとスケジュールテーブルの周期はタイマの周期（チック数）の整数倍にする必要があります。

方策 35c:　ジッタによる問題を回避するため、チックソースを扱う ISR はできる限り簡素化してください。ソフトウェアカウンタの場合は、API 関数 `IncrementCounter()`を呼び出してリターンするだけというのが理想的です。ハードウェアカウンタの場合は、`Os_AdvanceCounter_<CounterID>()`を呼び出し、必要に応じて自身をリトリガする処理を行います（『RTA-OS ユーザーガイド（RTA-OS User Guide）』を参照）。

方策 35d:　ハードウェアカウンタの場合、`Os_Cbk_Cancel_<CounterID>`、`Os_Cbk_Now_<CounterID>`、`Os_Cbk_Set_<CounterID>`と`Os_Cbk_State_<CounterID>`が正しく実装されているかをレビューする必要があります。これらのコールバックについては、『RTA-OS ユーザーガイド（RTA-OS User Guide）』で詳しく説明しています。小さい相対オフセットでアラーム／スケジュールテーブルが始動される可能性があるシステムにおいては、特に `Os_Cbk_Set_<CounterID>`の実装に注意が必要です。コールバックの起動がリクエストされてからコールバックに到達するまでの時間が、リクエストされているチック数より大きい場合は、コールバックがマッチのタイミングを逃したことを検知して適切なアクションを実行する必要があります。

想定 36:　AUTOSAR カウンタまたはスケジュールテーブルを駆動するために使用されるタイマはすべて、対応するカウンタまたはスケジュールテーブルのランタイム挙動を変更する必要がある場合に限り再設定されれるものとします。つまり、RTA-OS により生成された API 関数 `StartOS()`がアプリケーション内で呼び出された後は、タイマの再設定は行われないものとします。

責務 36:　‘tick’割り込みや‘advance’割り込みに使用されるハードウェアタイマがランタイムにおいて再設定されることがないようにする必要があります。

方策 36a:　システムコードをレビューして、明確に定義された処理のみがタイマの再設定を行うようにしてください。

方策 36b:　ハードウェアカウンタを使用する場合、RTA-OS の API 関数 `StartOS()`を呼び出した後は、適切な OS コールバック関数だけがタイマの再設定を行うようにしてくだ

さい。

想定 37: システム内に設定されているすべての AUTOSAR カウンタまたはスケジュールテーブルは、RTA-OS により生成された適切な API 関数によって正しいレートで駆動されるものとします。

責務 37: API 関数 IncrementCounter()／Os_AdvanceCounter_<CounterID>()またはそのバリアントが、システム内の予想外の場所から呼び出されないことを確認する必要があります。

方策 37a: API 関数 IncrementCounter()／Os_AdvanceCounter_<CounterID>()は、'tick'ソースからの割り込みを処理する ISR からのみ呼び出されるようにしてください。

方策 37b: システムコードをレビューして、API 関数 IncrementCounter()／Os_AdvanceCounter_<CounterID>()またはそのバリアントが、カウンタ／アラームおよびスケジュールテーブルの予期されたスケジューリング挙動のためにのみ使用されることを確認してください。

方策 37c: システムコードをレビューして、非トラステッドコードが API 関数 IncrementCounter()／Os_AdvanceCounter_<CounterID>()またはそのバリアントを予想外に呼び出さないように、カウンタおよびスケジュールテーブルのアクセス許可が正しく設定されていることを確認してください。

想定 38: システムが正しく動作するためにカウンタ／アラームまたはスケジュールテーブルが同期していることが重要な場合は、その同期はシステムにより保証されるものとします。

責務 38: システムがランタイムにおいてカウンタ／アラームとスケジュールテーブルの同期を維持することを確認する必要があります。

方策 38a: アラームを互いに同期させ続ける必要がある場合は、ランタイムにおいてそれらの始動や停止を行ってはいけないので、それらのアラームをスケジュールテーブルに変換することを検討してください。

方策 38b: ジッタによる問題を回避するため、同期的に起動する必要のあるすべてのタスクを 1 つのスケジュールテーブルで管理してください。

方策 38c: ランタイムのカウンタ／アラームまたはスケジュールテーブルの挙動を修正するには、必ず前もってシステムコードをレビューして、割り込みがディセーブルになっていることを確認してください。

想定 39: スケジュールテーブルの暗黙的同期化を行う場合、各スケジュールテーブルは必ず既知のカウンタ値で始動されるものとします。

責務 39: スケジュールテーブルを必ず既知のカウンタ値で始動する必要があります。

方策 39a: 暗黙的に同期化されるスケジュールテーブルの始動には、必ず API 関数 StartScheduleTableAbs()を使用してください。

想定 40: スケジュールテーブルの明示的同期化を行う場合は、スケジュールテーブルとそれらの同期化カウンタのモジュラス（カウンタ値がラップアラウンドする値）は必ず等しいものとします。

責務 40: 明示的に同期化される各スケジュールテーブルのモジュラスは、必ずその同期化カウンタのモジュラスと等しくする必要があります。

方策 40a: 同期化カウンタ（これは RTA-OS オブジェクトではありません）のドキュメントを参照してそのモジュラスを確認し、RTA-OS ツールを使用して、そのカウンタに関連

付けられているスケジュールテーブルの周期（duration）にその値を設定してください。

想定 41: スケジュールテーブルの明示的同期化を行う場合は、各スケジュールテーブルの駆動カウンタの分解能と同期化カウンタの分解能は必ず等しいものとします。

責務 41: 駆動カウンタの 1 チックとそれに関連付けられている同期化カウンタの 1 チックの時間の長さが等しいことを確認する必要があります。

方策 41a: 同期化カウンタ（これは RTA-OS オブジェクトではありません）のドキュメントを参照してその分解能を確認し、分解能が等しくなるように駆動カウンタを設定してください。

想定 42: スケジュールテーブルの明示的同期化を行う場合は、各スケジュールテーブルは、始動後、直ちに同期化されるものとします。

責務 42: スケジュールテーブルの正しい動作がシステムによってセットアップされるようする必要があります。

方策 42a: システムコードをレビューして、StartScheduleTableSynchron()に続いて必ず SyncScheduleTable()が呼び出されることを確認してください。

想定 43: スケジュールテーブルの切り替えを行う際に、'To'スケジュールテーブル（切り替え先のスケジュールテーブル）が特定された後に'From'スケジュールテーブル（切り替え元のスケジュールテーブル）は停止されないものとします。これにより、'To'スケジュールテーブルが SCHEDULETABLE_NEXT 状態のままとなる事態が回避されます。

責務 43: ランタイムにおいてスケジュールテーブルを切り替える処理においては、『RTA-OS ユーザーガイド（RTA-OS User Guide）』の例 10-7 のように、各 API 関数が正しい順序で使用されていることを確認する必要があります。

方策 43a: システムコードをレビューして、スケジュールテーブルの切り替え時には、'To'スケジュールテーブルが始動される前に'From'スケジュールテーブルが停止されることを確認してください。

想定 44: 拡張タスクが待ち状態のときに起動されることにより発生する可能性のある競合状態を防ぐため、すべての拡張タスクは自動起動される無限ループとして実装されるものとします。

責務 44: API 関数 ActivateTask()の呼び出しによって待ち状態の拡張タスクが起動されることのないようにする必要があります。

方策 44a: すべての拡張タスクは WaitEvent()呼び出しを実行する無限ループとして実装し、自動起動されるように設定してください。それにより、ActivateTask()で拡張タスクを起動する必要がなくなります。

想定 45: あるイベントがセットされた後にそのイベントに対応する拡張タスクが起動されることにより発生する可能性のある競合状態を防ぐため、すべての拡張タスクは自動起動される無限ループとして実装されるものとします。

責務 45: 休止状態（Suspended）の拡張タスクに対して API 関数 SetEvent()でイベントをセットすることがないようにする必要があります。

方策 45a: すべての拡張タスクは WaitEvent()呼び出しを行う無限ループとして実装し、自動起動されるように設定してください。それにより、拡張タスクが休止状態に入るのを防ぐことができます。

想定 46: システム内で受動的タイミング保護機能を使用する場合は、適切なハードウェアタイマがマイクロコントローラ上に存在し、RTA-OS により生成されたコードがこのハードウェアタイマをフリーランニングストップウォッチとして使用できるものとします。また、コールバック関数 Os_Cbk_GetStopwatch()が適切に実装され、これを利用して RTA-OS がタイマのカレント値を読み取ることができるものとします。

責務 46: OS の時間的側面を規制するために選択されているタイマが、コード生成時において、RTA-OS により算出された要件を満たすように設定されていることを確認する必要があります。

方策 46a: マイクロコントローラのリファレンスマニュアルを参照して、タイミング保護用に選択されているハードウェアタイマに、十分な分解能と範囲が提供されていることを確認してください。

方策 46b: タイミング保護の各バジェット値と、それらの値がタイマのチック値に丸められる処理を確認し、丸め効果に起因して、RTA-OS による誤検知がランタイムにおいて発生しないようにしてください。

想定 47: システム内で能動的タイミング保護機能を使用する場合は、適切なハードウェアタイマがマイクロコントローラ上に存在し、RTA-OS により生成されたコードがこのハードウェアタイマをフリーランニングストップウォッチとして使用できるものとします。

責務 47: タイマが使用可能であり、そのタイマを設定して、アプリケーションに設定された実行バジェットを守るための正しいタイミングで割り込みを生成するようにできることを確認する必要があります。

方策 47a: タイマの選択や、ハードウェア値の取得と比較を行う際には、マイクロコントローラのリファレンスマニュアルを参照して、正しい最小周期で割り込みを生成するように設定できることを確認してください。

想定 48: システム内で能動的タイミング保護機能を使用する場合は、コールバック関数 Os_Cbk_GetStopwatch()、Os_Cbk_SuspendTimeLimit()、Os_TimingFaultDetected()が適切に実装されていて、RTA-OS がタイマのカレント値を読み取ることができるものとします。

責務 48: 能動的保護のためのコールバック関数を適切に実装する必要があります。

方策 48a: 各コールバック関数のソースコードをレビューして、すべて正しく実装されていることを確認してください。

方策 48b: 意図的に実行バジェットに違反する試験プログラムを実装して実行し、保護フックが予想どおりに呼び出されることを検証してください。

メモリ保護

想定 49: 生成された RTA-OS コードがメモリ保護ユニットを直接設定することはないものとします。

責務 49: MPU のユーザー定義メモリが正しくプログラムされていることを確認する必要があります。OS タスクのコンテキスト切り替え時にこれを変更する必要がある場合は、すべてのリプログラミングに正しいメモリ位置とレジスタ設定が使用されることを確認する必要があります。

方策 49a: システムコードをレビューして、コールバック関数 Os_Cbk_SetMemoryAccess()があらゆる状況において MPU のプログラミングを正しく実行することを確認してください。

方策 49b: RTA-OS から提供される例に従って、スタックポインタと MPU 領域が正しくアラインメントされるようにしてください。

想定 50: 生成される RTA-OS はトラステッドコードであり、与えられた特権を他のすべてのトラステッドコードと共有します。

責務 50: システム内のどのトラステッドコードも、システムの安全性を低下させるアクションを実行しないようにする必要があります。

方策 50a: システムコードをレビューし、ランタイムにおいてどのトラステッドコードも、RTA-OS が依存するハードウェア設定を変更しないことを確認してください。

想定 51: トラステッド OS アプリケーションがトラステッド関数をエクスポートする場合は、その関数の中に、非トラステッドオブジェクトが特権機能にアクセスすることを許すコードは含まれていないものとします。たとえば、トラステッド関数が「バックドア」として機能してしまうと、非トラステッドタスクが特権ペリフェラルレジスタのリプログラミングを行うことを許してしまう可能性があります。

責務 51: トラステッド OS アプリケーションによりエクスポートされるすべてのトラステッド関数は、非トラステッドコードから呼び出された際に、システムの安全性を低下させるアクションは実行できないようにする必要があります。

方策 51a: システムコードをレビューして、マイクロコントローラの特権モードで実行されるすべてのトラステッド関数について、システムの安全性を低下させないことを確認してください。

想定 52: OS アプリケーションは、システム機能の各領域間の境界を守るために用いられるものです。ある OS アプリケーション内のオブジェクトが別の OS アプリケーション内のオブジェクトにアクセスすることを許すと、保護の脆弱性が生じてしまう可能性があります。

責務 52: OS アプリケーション間のすべてのアクセス許可設定が、システムの安全性を低下させる可能性のあるオブジェクト操作によって予想外の副作用を招くことのないようにする必要があります。

方策 52a: OS アプリケーション間のアクセス許可は、すべて禁止してください。

方策 52b: OS アプリケーション間のアクセス許可が必要な場合は、オブジェクト操作によって発生する可能性のあるすべての副作用を慎重に分析して、それらの副作用がシステムの安全性を劣化させないことを確認してください。

想定 53: アラームコールバック関数が SC2、SC3、SC4 で使用される場合は、安全な機能だけが提供されるものとします。

責務 53: SC2、SC3、SC4 でアラームコールバック関数を使用する際に、システムの安全性を劣化させる副作用が発生しないようにする必要があります。

方策 53a: SC2、SC3、SC4 でアラームコールバック関数を使用することを許可する RTA-OS コンフィギュレーションオプションの使用を禁止してください。注記: これは AUTOSAR のデフォルトの挙動です。

方策 53b: SC2、SC3、SC4 でアラームコールバック関数を使用することを許可する RTA-OS コンフィギュレーションオプションを使用する必要がある場合は、コードをレビューして、安全上重大な副作用が生じる可能性が全くないことを確認してください。

コードとデータの破壊

想定 54: 生成された RTA-OS コードの完全性は、開発プロセス内の下流ツールによってシステムに統合された場合も低下することはないものとします。

責務 54: どの下流ツールも、生成された OS の正当性を低下させないようにする必要があります。

方策 54a: RTA-OS により生成されたコードをビルドしてシステム用ライブラリを作成する際に使用するツール（コンパイラ、アセンブラ、リンカ）の使用についてレビューしてください。コンパイラオプションのうち、必須のものと使用禁止のものに関する情報は、各ターゲット用の『RTA-OS Port Guide』に記述されています。

想定 55: システムメモリ内で検知された故障はすべて、生成された RTA-OS とは無関係に処理されるものとします。

責務 55: システムメモリの破壊を検知するために必要なテストとチェックは、すべて RTA-OS とは無関係に行われ、エラーが見つかった際にはグレースフルデグラデーション（性能の制限によるシステムの維持）またはシステムシャットダウンが行われるようにする必要があります。

方策 55a: OS に SC3 または SC4 を設定してメモリ保護機能を実装し、非トラステッドコードには実際に必要な最低限のメモリ領域にしかアクセスさせないようにしてください。

方策 55b: OS に SC3 または SC4 を設定して RTA-OS 拡張ステータスを実装し、無効な API パラメータや範囲外のパラメータ値がないか調べるためのあらゆるランタイムチェックが確実に行われるようにしてください。

プログラムフローのエラー

想定 56: 生成された RTA-OS 内の API は、すべて正しく使用されることとします。

責務 56: RTA-OS の API は、すべて正しく使用する必要があります。

方策 56a: 『RTA-OS リファレンスガイド（RTA-OS Reference Guide）』を参照してシステムコードをレビューし、すべての API 呼び出しと、そのパラメータと戻り値が正しく使用されていることを確認してください。

方策 56b: 拡張ビルドステータスと AUTOSAR サービス保護を有効にして、アプリケーションのすべてのパスカバレッジ試験において RTA-OS のエラーフック（ErrorHook）関数の呼び出しが実行されないことを確認してください。

想定 57: 生成された RTA-OS コードのランタイムエラー検知機能は、正しく効果的に使用されるものとします。

責務 57: RTA-OS のランタイムエラー検知機能は、適切に使用する必要があります。

方策 57a: SC3 または SC4 を設定して、拡張ステータスの RTA-OS によってすべてのランタイムエラーチェックが確実に行われるようにしてください。

方策 57b: システムソースコードをレビューして、API からの戻り値がすべてチェックされ、適切なエラー処理が実装されていることを確認してください。

方策 57c: 実装されたエラーフックをレビューし、すべてのエラーコードが検知されて処理されることを確認してください。RTA-OS ツールでサンプルのエラーフック関数を生成することができるので、参考にしてください。

想定 58: ハードウェア障害検知は、生成された RTA-OS とは無関係に処理されるものとします。

責務 58: ハードウェアの障害を検知するために必要なテストとチェックは、すべて RTA-OS とは無関係に行われ、障害が見つかった際にはグレースフルデグラデーション（性能の制限によるシステムの維持）またはシステムシャットダウンが行われるようにする必要があります。

方策 58a: SC3 または SC4 を使用してメモリ保護機能を実装し、MPU を使用して、非トラステッドコードがマイクロコントローラペリフェラルにアクセスすることを防いでください。

方策 58b: ロックステップ方式のコアを搭載するマイクロコントローラを使用して、ランタイムにおける予想外のプログラムフローを検知するメカニズムを装備してください。

9.2 RTA-OS の実装に関する想定

ETAS は、生成された OS に関する以下の特性は、「安全性を証明する議論」として一般的に信頼できるものであると想定しています。

これらの想定事項は、本書や各手引書において正当性と安全性が議論される OS の各側面を表しています。詳細は第 10 章を参照してください。

AUTOSAR OS を使用するアプリケーションのための安全コンセプトは、生成された OS の以下の特性により成立するものです。

想定 59: RTA-OS ツールにより生成された OS は、AUTOSAR OS のすべての機能を適切な SWS ドキュメントに定められているとおりに実装します。正しくない動作はすべてバグによるものであり、ETAS がそれを再現できる場合は修正されます。

想定 60: RTA-OS ツールにより生成された OS は、基盤をなすマイクロコントローラの挙動や性能の試験／監視を一切行いません。

想定 61: RTA-OS ツールにより生成された OS は、スケジューリングポイントにおいて、固定優先度プリエンプティブスケジューリングのルールに従い、システム内でレディ状態になっている最高優先度のタスク／ISR を他のどのタスクよりも優先して実行します。

想定 62: RTA-OS ツールにより生成された OS は、カテゴリ 2 ISR を、即時継承・優先度上限プロトコルに従い、システム内の他のどのタスクよりも優先して直ちに実行します。

想定 63: RTA-OS ツールにより生成された OS は、カテゴリ 1 ISR を、システム内のどのカテゴリ 2 ISR またはタスクよりも優先して実行します。

想定 64: リソースオブジェクトがロックされているとき、RTA-OS ツールにより生成された OS は、そのリソースをロックしているタスクまたは ISR を、そのリソースを使用できる他のタスクやカテゴリ 2 ISR がプリエンプトできないようにします。

想定 65: スピンロックオブジェクトがロックされているとき、RTA-OS ツールにより生成された OS は、ロックを行っているタスクまたは ISR を、そのスピンロックを使用できる他のタスクやカテゴリ 2 ISR がプリエンプトできないようにします。

想定 66: システムにタイミング保護が設定されている場合、RTA-OS ツールにより生成された OS は、設定されている実行バジェットを超過したコードを検知し、保護フックによってシステムにエラーを通知します。

想定 67: システムにタイミング保護が設定されている場合、RTA-OS ツールにより生成された OS は、設定されたアライバル間隔よりも早くアライバルが発生したオブジェクトを検知し、保護フックを通じてシステムにエラーを通知します。

想定 68: システムにメモリ保護が設定されている場合、RTA-OS ツールにより生成された OS は、非トラステッドコードを実行する直前にコールバック関数 `Os_Cbk_SetMemoryAccess()`を実行します。

10 生成された RTA-OS コードの安全性を証明する議論

RTA-OS ツールにより生成された OS コードは、ユーザーの安全関連 ECU システム内の「ソフトウェアユニット」（ソフトウェアの単体部品）であり、これを ECU システムのセーフティケースに統合することはユーザーの責務となります。本章では、セーフティケースを構築する際に役立つ情報を紹介します。さらに詳細な情報は、第 11 章に記載された情報を参照して ETAS までお問い合わせください。

10.1 故障モード

生成された RTA-OS の挙動は外的要因に影響される可能性があるため、いくつかの故障モードが考えられます。本項の情報は、生成された RTA-OS に発生した故障を特定するのに役立つ可能性があります。

生成された RTA-OS の通常動作は AUTOSAR OS SWS [1]に定められているとおりです。ただし、生成されたコードは必ずハードウェア環境内で実行されることになるため、ECU システムによる API 呼び出しを受けることになります。

生成された RTA-OS コードに関連する故障モードには以下のものがあります。

1. RTA-OS コードにバグが含まれている
2. RTA-OS コード内の API 関数が誤った方法で呼び出される
3. RTA-OS コードが使用するベクタテーブルが初期化されない
4. RTA-OS コードが使用するデータ構造またはレジスタが初期化されない
5. RTA-OS コードが想定しているマルチコアスタートアップシーケンスに従ってスタートアップが行われない
6. RTA-OS コードが、適合しないハードウェア上で実行される

表 10-1 は、RTA-OS ツールの各故障モードが発生した際にそれを検知するための参考情報をまとめたものです。「発生確率」と「影響度」の値は ETAS の経験に基づいて設定されたものなので、ユーザーの ECU システムについてはこれとは別の値を用いることができます。いずれにしても、発生確率と影響度の高い故障モードについて集中的に試験するようにしてください。

#	故障モード	考えられる故障状態	発生確率	影響度	確認の方法
1	1	RTA-OS コードが予想外の挙動を示す	低	高	ECU アプリケーションが予想どおりの挙動を示しません。
2	2	RTA-OS の API 関数が E_OK 以外のエラーコードを返すか、予想外の挙動を示す	高	高	RTA-OS 設定時に拡張ステータスを指定し、API エラーチェックがフルに行えるようにしてください。すべての API が正しく呼び出されていることが確認できた後は、標準ステータスに切り替えてください。標準ステータスでは一部の API エラーチェックしか行われません。 エラーフックを使用して、API エラーを検出して対処してください。 すべての API からの戻り値をチェックし、適切に対処してください。

#	故障モード	考えられる故障状態	発生確率	影響度	確認の方法
3	3	予想される割り込みを RTA-OS コードが検知しない	高	高	API 関数 Os_InitializeVectorTable()が各ターゲットの『RTA-OS Port Guide』に記述されているとおりに呼び出されているかを確認してください。ユーザーが RTA-OS ツールを使用してベクタテーブルを生成していない場合でも、生成された RTA-OS コードはランタイムに利用するレジスタを初期化しなければならないので、必ずこの API 呼び出しを行う必要があります。
4	4	データがゼロに設定されていないか、または間違った値に初期設定されていることにより、アラームまたはスケジュールテーブルが予想外のタイミングで起動される	低	高	RTA-OS コードのデータの一般的な初期化処理（ゼロ設定など）は、C スタートアップコードで行われます。トレースツール（デバッガや RTA-TRACE など）を使用して、設定されたタイミングでアラームやスケジュールテーブルが処理されるかを確認してください。
5	4	レジスタが初期化されていないため、ECU アプリケーションが起動された直後にクラッシュする	低	高	すべてのマイクロコントローラレジスタ（スタックポインタなど）が C スタートアップコードまたはユーザーのターゲット初期化コードによって適切に初期化されるかを確認してください。デバッガを使用して、ランタイムにおいてレジスタの内容が正しいことを検証してください。
6	5	RTA-OS スタートアップシーケンスが守られないため、スタートアップのほぼ直後にマルチコアアプリケーションがデッドロックするように見える	高	高	デバッガを使用して、すべてのマイクロコントローラコアが Idle コールバックに到達するかを確認してください。『RTA-OS ユーザーガイド（RTA-OS User Guide）』（参考文書**[10]**）の第 22 章に記述されている、API 関数を用いたマルチコアの始動シーケンスが確実に実行されるようにしてください。
7	6	マイクロコントローラのアーキテクチャが予想されたものと異なるため、ECU アプリケーションがクラッシュする	低	高	デバッガを使用して、アプリケーションが予想どおりに実行されるかを確認してください。生成された RTA-OS コードは、特定のマイクロコントローラアーキテクチャ用に最適化されています。ECU のマイクロコントローラアーキテクチャ（割り込みコントローラ、クロスバーなど）が異なると、未定義の挙動を示します。

表 10-1: RTA-OS ツールの故障を検出するための提案

10.2 開発プロセス

RTA-OS により生成されるコードの設計は、6.2 項と 7.6.2 項で説明されている ETAS 製品エンジニアリングプロセス（PEP）に従って行われています。自動トレースツールの使用に関しても同じ議論が適用され、生成されたコードの試験を行ってすべての要件が満足されていることを確認します。

10.3 RTA-OS により生成されたコードの安全性に関与する特性

生成された OS コードには、ECU アプリケーション用のセーフティケースを構築するために利用できるいくつかの特性があります。

- 完全に静的に定義された OS: コード生成の時点ですべてのオブジェクトが確定されるので、それらをすべて静的にアロケートすることができます。
- ランタイムにおいて OS が動的メモリアロケーションを行うことはありません。
- スケジューリングは純粋に優先度ベースで行われます。
- 即時継承・優先度上限プロトコル（別名: スタックベースプロトコル）に従ってロックされるバイナリセマフォ（「リソース」と呼ばれます）により、ブロッキング時間が最小限に抑えられます。
- 生成された OS コードは MISRA-C:2004 指針[5]に準拠しています。これは、C 言語に関して国際的に認められている一連のコーディング指針で、この指針から逸脱する部分については、『RTA-OS MISRA Deviation Report（RTA-OS MISRA 逸脱レポート）』に説明されています。
- 生成された OS コードには、明確に定義されたネームスペースが使用されています。
- 生成された OS コードには明確に定義された命名規則が使用されています。OS が内部的に使用するすべての名前は、入力された XML 名から導出されるので、生成された OS コード内のオブジェクトの有無を‘grep’や‘diff’などのツール [8]により容易に機械的にチェックできます。
- RTA-OS のすべての API は内部メカニズムにより保護されているので、データの完全性が保証されます。アプリケーションコードが API の使用に関する規約を順守する限り、通常動作中に OS データ構造が破壊されることはありません。

[8] ほとんどのOS（LinuxやWindowsなど）で同様のファイル・文書処理用ツールを使用できます。

11 お問い合わせ先

ETAS本社

ETAS GmbH

Borsigstraße 14	Phone:	+49 711 3423-0
70469 Stuttgart	Fax:	+49 711 3423-2106
Germany	WWW:	www.etas.com

その他のお問い合わせ先

お近くの営業所やサポート窓口についての情報は、ETAS ホームページをご覧ください。

各国支社	WWW:	www.etas.com/ja/contact.php
テクニカルサポート	WWW:	www.etas.com/ja/hotlines.php

11.1 テクニカルサポート

サポート契約をに基づき RTA-OS をお使いいただいているユーザーの方には、テクニカルサポートをご提供させていただいております。サポート契約を結ばれていないお客様は営業窓口（上記参照）までお問い合わせください。

テクニカルサポートをご利用いただく際には、E メールでご連絡いただくのが最も確実な方法です。製品の技術的な問題やご不明の点は、下記アドレス宛の E メールでお気軽にお問い合わせください。

RTA_Hotline.ETAS_MB_YH@etas.com （日本国内のホットライン窓口）

テクニカルサポートチームに直接お話しになりたい場合は、ホットライン窓口の下記番号までお電話ください。

045-222-0951 （日本国内のホットライン窓口）

お電話での受付時間は、平日の営業時間内（9:00～18:00）です。

どちらの場合も、以下の情報をご提供いただけると問題解決に役立ちます。

- お客様のサポート契約番号
- ご使用の ETAS ツールとそのバージョン
- ご使用のコンパイラツールチェーンとそのバージョン
- ユーザーデータ（`.xml`、`.arxml`、`.rtaos`、`.stc` ファイル）
- エラーを再現させる手順に関する説明
- システムからエラーメッセージが出力された場合は、そのメッセージ
- `Diagnostic.dmp` というファイルが生成された場合は、そのファイル

12 参考文書

[1] AUTOSAR Specification of Operating System v5.0.0 R4.0 Rev 3

[2] AUTOSAR Specification of ECU Configuration v3.2.0 R4.0 Rev 3

[3] AUTOSAR Specification of Watchdog Manager v2.2.0 R4.0 Rev 3

[4] ISO 26262:2011(E) Road vehicles – Functional safety

[5] MISRA-C:2004 Guidelines for the use of the C language in critical systems

[6] OSEK/VDX Operating System Specification 2.2.3

[7] RTA-OS Analysis Visualizer User Guide 10565-AVUG-5.1.0 EN-08-2012

[8] RTA-OS Getting Started Guide 10565-GS-5.1.0 EN-08-2012
（日本語版タイトル: RTA-OS 入門ガイド）

[9] RTA-OS Reference Guide 10565-RG-5.1.0 EN-08-2012
（日本語版タイトル: RTA-OS リファレンスガイド）

[10] RTA-OS User Guide 10565-UG-5.1.0 EN-08-2012
（日本語版タイトル: RTA-OS ユーザーガイド）

第 4 部（付録）RTA-OS の統合に関するチェックリスト

13 付録の概要

13.1 はじめに

この付録（第 4 部）は、RTA-OS ツール、および生成された RTA-OS コードを、安全関連の ECU システムに統合する際に考慮する必要のある一般的な課題のチェックリストになっています。チェックリストは、RTA-OS が所定のターゲットハードウェア用に正しく設定されて統合されていることを示すための一般的指針として使用されることを目的としています。従って、チェックリスト内の項目のうち、結果が「Yes」となったものは、生成された RTA-OS コードを使用した ECU アプリケーションのクオリティケース（品質証明）の一環として利用することができます。また、開発プロセスでの RTA-OS ツールの使用や生成された RTA-OS コードの ECU システムへの統合に関してのレビューと試験を行う際にも、このチェックリストを活用することができます。

このチェックリストは、ECU システム自体の試験やデバッグのために作成されたものではありません。ECU システムの設計と開発のプロセスには完全な機能試験スイートが不可欠です。

警告！

ユーザーは、選択されている ASIL について ISO26262 により要求されているレビュー、カバレッジ、試験を実行する必要があります。

RTA-OS の API の正しい使用方法については、このチェックリストではカバーされていません。チェック対象のアプリケーションについては、すでにコンパイルとリンクのエラーがないことが想定されていて、すべての API 関数が正しい構文で呼び出されていることが前提となります。

警告！

このチェックリストのすべての項目の結果が「yes」となった場合でも、生成された RTA-OS コードを安全に使用できることが保証されるわけではありません。生成された RTA-OS コードが所定の安全目標を実現するかどうかは、ユーザーが責任をもって確認する必要があります。

13.2 チェックリストの使い方

以降に掲載されているチェックリストは、RTA-OS のコンフィギュレーション設定と統合に関しての正当性と一貫性を実証する際に役立てていただくことを目的としています。チェックリストは各章ごとに、RTA-OS ツール、または生成された RTA-OS コードに接続して機能するアプリケーションのソースコードの一部に関連しています。そのため、チェックリストに記入する際には、RTA-OS ツール、アプリケーションの ARXML コンフィギュレーションファイル、およびアプリケーションのソースコードへのアクセスが必要になります。検討中のターゲットの『RTA-OS Port Guide』の内容を理解しておくと、チェックリストの記入に役立ちます。

チェックリスト内の各項目（質問事項）は「Yes」／「No」の欄にチェックを入れるようになっていて、コメント欄も用意されています。たとえば、アプリケーションの資料に推奨されているコンパイラオプションが使用されていない場合は、その理由や、それによってコードに期待される効果をコメント欄に記入してください。

チェックリストへの記入時にサポートが必要な場合は、ETAS のコンサルタントサービスをご利用いただけます。ETAS は RTA-OS の使用に関するレビューをお手伝いできます。

14 RTA-OS ツールの統合に関するチェックリスト

本章には RTA-OS ツールで設定される ECU アプリケーションのコンフィギュレーションに関するチェック項目が掲載されています。

ターゲットバリアント

1.	システム内で使用されるマイクロコントローラおよびコンパイラについて、RTA-OS ツールの OS Configuration > General > Target において、ターゲットの正しい情報（名前、バージョン、バリアント）が設定されていますか？	Yes	No
コメント			

タスクレジスタセットの使用

2.	カスタマイズされたコンテキストの保存と復元のために RTA-OS レジスタセットを利用するすべてのタスクが、そのタスクのコンフィギュレーション内でその旨を宣言していますか？	Yes	No
コメント			

スタックアロケーション

3.	アプリケーションが拡張タスクを使用する場合、コンフィギュレーション内にすべてのタスクとカテゴリ 2 ISR のスタックサイズが設定されていますか？	Yes	No
コメント			

WaitEvent()のスタックアロケーション

4.	アプリケーションが拡張タスクを使用する場合、すべての拡張タスクについて設定されている‘Wait-Stack size’のアロケーションは、拡張タスクが `WaitEvent()`を呼び出す時点のスタックコンテキストを保持するのに十分ですか？	Yes	No
コメント			

レジスタセットの使用

5.	コンテキスト切り替え時にレジスタセットを使用して CPU レジスタの保存と復元を行う必要があるすべての ISR が、その ISR のコンフィギュレーション内でその旨を宣言していますか？	Yes	No
コメント			

リソースのコンフィギュレーション

6.	リソースを使用するすべてのタスク／ISR が、RTA-OS ツールの OS Configuration > Resources に設定されていますか？	Yes	No
コメント			

イベントのセット

7.	イベントとそれを待つタスクの完全なマッピングが、RTA-OS ツールの OS Configuration > Events に設定されていますか？	Yes	No
コメント			

イベントの待機

8.	ソースコード内のタスクによって待機されるすべてのイベントが、RTA-OS ツール内に設定されていますか？	Yes	No
コメント			

15 生成された RTA-OS コードの統合に関するチェックリスト

本章には、生成された RTA-OS コードをユーザーの ECU システムコード内で使用することに関するチェック項目が掲載されています。

コンパイラバージョン

9.	アプリケーションは、『RTA-OS Port Guide』に明記されている推奨コンパイラバージョンでコンパイルされていますか？	Yes	No
コメント			

アプリケーションコード用の推奨コンパイラオプション

10.	アプリケーションは、『RTA-OS Port Guide』に明記されている推奨コンパイラオプションを使用してコンパイルされていますか？	Yes	No
コメント			

アセンブラバージョン

11.	アプリケーションは、『RTA-OS Port Guide』に明記されている推奨アセンブラバージョンでアセンブルされていますか？	Yes	No
コメント			

アプリケーションコード用の推奨アセンブラオプション

12.	アプリケーションは、『RTA-OS Port Guide』に明記されている推奨アセンブラオプションを使用してアセンブルされていますか？	Yes	No
コメント			

リンカバージョン

13.	アプリケーションは、『RTA-OS Port Guide』に明記されている推奨リンカバージョンでリンク／ロケートされていますか？	Yes	No
コメント			

アプリケーションコード用の推奨リンカオプション

14.	アプリケーションは、『RTA-OS Port Guide』に明記されている推奨リンカオプションを使用してリンク／ロケートされていますか？	Yes	No
コメント			

データセクションの初期化

15.	C スタートアップコードはアプリケーションのすべてのデータセクションを正しく初期化しますか？	Yes	No
コメント			

スタックスペース

16.	アプリケーション用に十分なスタックスペースがリンカによりアロケートされていますか？	Yes	No
コメント			

レジスタの初期化

17.	必要なすべてのマイクロコントローラレジスタは『RTA-OS Port Guide』に明記されているとおりに正しく初期化されますか？	Yes	No
コメント			

StartOS()呼び出し

18.	StartOS()は OS_MAIN()関数内から 1 回だけ呼び出されますか？	Yes	No
コメント			

フックルーチンの提供

19.	RTA-OS ツールの OS Configuration > General に設定されているフックルーチンは、アプリケーションソースコード内で以下のように正しく定義されていますか？ FUNC(void, OS_CALLOUT_CODE) StartupHook(void); FUNC(void, OS_CALLOUT_CODE) ShutdownHook(StatusType s); FUNC(void, OS_CALLOUT_CODE) PreTaskHook(void); FUNC(void, OS_CALLOUT_CODE) PostTaskHook(void); FUNC(void, OS_CALLOUT_CODE) ErrorHook(StatusType e); FUNC(void, OS_CALLOUT_CODE) Os_Cbk_StackOverrunHook(Os_StackSizeType Overrun, Os_StackOverrunType Reason) FUNC(ProtectionReturnType, OS_CALLOUT_CODE) ProtectionHook(StatusType FatalError)	Yes	No
コメント			

StartupHookが使用するAPI

20.	StartupHook は以下の API 関数のみを使用していますか？ ActivateTask() DisableAllInterrupts() EnableAllInterrupts() GetActiveApplicationMode() GetApplicationID() GetApplicationState() GetCoreID() Os_GetStackSize() Os_Get_Stack_Value() Os_Get_ActivationTime() Os_GetVersionInfo() ResumeAllInterrupts() ResumeOSInterrupts() SetAbsAlarm() SetRelAlarm() SetScheduleTableAsync() ShutdownAllCores() ShutdownOS() StartCore() StartNonAutosarCore() StartScheduleTableAbs() StartScheduleTableRel() StartScheduleTableSynchron() SuspendAllInterrupts() SuspendOSInterrupts() SyncScheduleTable()	Yes	No
コメント			

ShutdownHookが使用するAPI

21.	ShutdownHook は以下の API 関数のみを使用していますか？ DisableAllInterrupts() EnableAllInterrupts() GetActiveApplicationMode() GetApplicationID() GetApplicationState() GetCoreID() Os_GetStackSize() Os_Get_Stack_Value() Os_Get_ActivationTime() Os_GetVersionInfo() ResumeAllInterrupts() ResumeOSInterrupts() SuspendAllInterrupts() SuspendOSInterrupts()	Yes	No
コメント			

PreTaskHookが使用するAPI

22.	PreTaskHook は以下の API 関数のみを使用していますか？ DisableAllInterrupts() EnableAllInterrupts() GetActiveApplicationMode() GetApplicationID() GetApplicationState() GetCoreID() GetEvent() GetTaskID() GetTaskState() GetActiveApplicationMode() GetAlarm() GetAlarmBase() GetEvent() Os_GetStackSize() Os_Get_Stack_Value() Os_Get_ActivationTime() Os_GetVersionInfo() ResumeAllInterrupts() ResumeOSInterrupts() SuspendAllInterrupts() SuspendOSInterrupts()	Yes	No

PostTaskHookが使用するAPI

23.	PostTaskHook は以下の API 関数のみを使用していますか？ DisableAllInterrupts() EnableAllInterrupts() GetActiveApplicationMode() GetAlarm() GetAlarmBase() GetApplicationID() GetApplicationState() GetCoreID() GetEvent() GetTaskID() GetTaskState() Os_GetStackSize() Os_Get_Stack_Value() Os_Get_ActivationTime() Os_GetVersionInfo() ResumeAllInterrupts() ResumeOSInterrupts() SuspendAllInterrupts() SuspendOSInterrupts()	Yes	No
コメント			

ErrorHookが使用するAPI

24.	ErrorHook は以下の API 関数のみを使用していますか？ CheckISRMemoryAccess() CheckObjectAccess() CheckObjectOwnership() CheckTaskMemoryAccess() DisableAllInterrupts() EnableAllInterrupts() GetActiveApplicationMode() GetAlarm() GetAlarmBase() GetApplicationID() GetApplicationState() GetCoreID() GetEvent() GetISRID() GetTaskID() GetTaskState() Os_GetISRMaxExecutionTime() Os_GetISRMaxStackUsage() Os_GetStackSize() Os_GetStackValue() Os_GetTaskActivationTime() Os_GetTaskMaxExecutionTime() Os_GetTaskMaxStackUsage() Os_GetVersionInfo() Os_ResetISRMaxExecutionTime() Os_ResetISRMaxStackUsage() Os_ResetTaskMaxExecutionTime() Os_ResetTaskMaxStackUsage() ResumeAllInterrupts() ResumeOSInterrupts() ShutdownAllCores() ShutdownOS() SuspendAllInterrupts() SuspendOSInterrupts()	Yes	No
コメント			

ProtectionHookが使用するAPI

25.	ProtectionHook は以下の API 関数のみを使用していますか？ CheckISRMemoryAccess() CheckObjectAccess() CheckObjectOwnership() CheckTaskMemoryAccess() DisableAllInterrupts() EnableAllInterrupts() GetApplicationID() GetApplicationState() GetCoreID() GetISRID() GetTaskID() Os_GetStackSize() Os_GetStackValue() Os_GetTaskActivationTime() Os_GetTaskMaxStackUsage() Os_GetVersionInfo() ResumeAllInterrupts() ResumeOSInterrupts() SuspendAllInterrupts() SuspendOSInterrupts()	Yes	No
コメント			

StackOverrunHookが使用するAPI

26.	StackOverrunHook は以下の API 関数のみを使用していますか？ CheckObjectAccess() CheckObjectOwnership() DisableAllInterrupts() EnableAllInterrupts() GetActiveApplicationMode() GetAlarmBase() GetApplicationID() GetApplicationState() GetCoreID() GetTaskID() Os_GetStackSize() Os_GetStackValue() Os_GetTaskActivationTime() Os_GetVersionInfo() ResumeAllInterrupts() SuspendAllInterrupts()	Yes	No
コメント			

TimeOverrunHookが使用するAPI

27.	TimeOverrunHook は以下の API 関数のみを使用していますか？ CheckObjectAccess() CheckObjectOwnership() DisableAllInterrupts() EnableAllInterrupts() GetActiveApplicationMode() GetAlarm() GetAlarmBase() GetApplicationID() GetApplicationState() GetCoreID() Os_GetISRMaxExecutionTime() Os_GetISRMaxStackUsage() Os_GetStackSize() Os_GetStackValue() Os_GetTaskActivationTime() Os_GetTaskMaxExecutionTime() Os_GetTaskMaxStackUsage() Os_GetVersionInfo() Os_ResetISRMaxExecutionTime() Os_ResetISRMaxStackUsage() Os_ResetTaskMaxExecutionTime() Os_ResetTaskMaxStackUsage() ResumeAllInterrupts() ShutdownOS() SuspendAllInterrupts()	Yes	No
コメント			

OSコールバック

28.	アプリケーションが拡張ステータスを使用する場合、各 OS コールバックルーチンが以下の例のように正しくソースコード内に定義されていますか？ FUNC(void, OS_CALLOUT_CODE) Os_Cbk_TimeOverrunHook(Os_StopwatchTickType Overrun); FUNC(Os_StopwatchTickType, OS_CALLOUT_CODE) Os_Cbk_GetStopwatch(void);	Yes	No
コメント			

エラー処理APIの保護

29.	高度なエラーロギング（Advanced Error Logging）が設定されていない場合は、エラー処理 API（OSErrorGetServiceID()）の呼び出しが行われないように、適切な#define 文によって保護されていますか？
	
	`#if OS_EXTENDED_STATUS`
	`  OSErrorGetServiceID();`
	`#endif`
		Yes	No

コメント			

タイミングAPIの保護

30.	アプリケーションが時間監視 API を使用する場合、Time Monitoring（時間監視）が 設定されていなければその API が使用されないように、以下の例のような適切なマクロによって保護されていますか？
	
	`#ifdef OS_TIME_MONITORING`
	`  GetLargestExecutionTime(MyTask, WCET_MyTask);`
	`  GetExecutionTime(WCET_to_this_point);`
	`#endif`
		Yes	No

コメント			

TASKマクロ

31.	すべてのタスクのエントリ関数に、以下の例のように TASK()マクロが使用されていますか？
	
	`TASK(myTask) {`
	`}`
		Yes	No

コメント			

正しいターミネーション

32.	RTA-OS に‘Fast Terminate（高速ターミネート）’による最適化が選択されている場合、アプリケーション内のすべてのタスクは必ずそのエントリ関数の中でターミネート（つまり API 関数 TerminateTask()または ChainTask()の呼び出し）を行いますか？ 注記: ‘Fast Terminate’による最適化を行うには、すべてのタスクが必ずそのエントリ関数の中でターミネートする必要があります。これにより、RTA-OS はタスク実行に必要なスタックメモリを最適化できます。この最適化が無効の場合、各タスクは、エントリ関数から呼び出される関数内を含むタスクボディ内のどこでもターミネートできます。	Yes	No
コメント			

API関数 Schedule()

33.	RTA-OS ツールの OS Configuration > General > Optimizations において‘Disallow Schedule()’による最適化が選択されている場合、どのタスクも API 関数 Schedule()を使用しない、ということが確認できますか？	Yes	No
コメント			

上方起動

34.	RTA-OS ツールの OS Configuration > General > Optimizations において‘Disallow Upwards Activation’による最適化が選択されている場合、どのタスクも自分より優先度の高いタスクを直接起動することがない、ということが確認できますか？ この「直接起動」には、API 関数 ActivateTask()または ChainTask()を呼び出す、イベントをセットする、カウンタを進める、または他のタスクスケジューリングメカニズムを使用する、といったことが含まれます。 注記: この最適化が有効の場合、RTA-OS はランタイムにおいてこのようなタスク起動を「自動的に（silently）」回避します。	Yes	No
コメント			

拡張タスクの実装

35.	アプリケーションが拡張タスクを使用する場合、休止状態（suspended）の拡張タスクに対して SetEvent()が呼び出されるのを防ぐため、すべての拡張タスクが WaitEvent()の呼び出しを実行する無限ループとして実装されていますか？	Yes	No
コメント			

CAT1_ISRマクロ

36.	すべてのカテゴリ 1 ISR が、以下の例のように CAT1_ISR マクロを使用して宣言されていますか？
	
	`CAT1_ISR(myCat1ISR) {`
	`}`
		Yes	No

コメント			

カテゴリ1 ISRのターミネーション

37.	すべてのカテゴリ 1 ISR が、適切な‘return from interrupt’（またはそれと同等の）命令シーケンスを実行していますか？	Yes	No
コメント			

ISRマクロ

38.	すべてのカテゴリ 2 ISR が、以下の例のように ISR マクロを使用して宣言されていますか？
	
	`ISR(myCat2ISR) {`
	`}`
		Yes	No

コメント			

カテゴリ2 ISRのターミネーション

39.	カテゴリ 2 ISR の中で‘return from interrupt’（またはそれと同等の）命令シーケンスが実行されない、ということが確認できますか？ 注記: RTA-OS は、割り込みハードウェア制御をすべて割り込みラッパー関数内で実行します。	Yes	No
コメント			

ベクタのマッピング

40.	すべての ISR がターゲットマイクロコントローラ上の正しい割り込みベクタにマッピングされていますか？	Yes	No
コメント			

すべてのソース用の割り込みハンドラ

41.	予想外の割り込みが誤って処理されるのを防ぐため、許可されている（enable になっている）割り込みソースごとに割り込みハンドラが提供されていますか？	Yes	No
コメント			

割り込み優先度レベル

42.	ターゲット専用に生成された API 関数 Os_InitializeVectorTable()を呼び出すことにより、割り込み優先度レベルをコンフィギュレーションに基づいて正しく設定していますか？ 注記: Os_InitializeVectorTable()についての詳細は各ターゲット用『RTA-OS Port Guide』に記述されています。	Yes	No
コメント			

カテゴリ1 ISRによるAPIの使用

43.	カテゴリ 1 ISR は以下の API 関数のみを使用していますか？ DisableAllInterrupts() EnableAllInterrupts() GetCoreID() Os_GetStackSize() Os_GetStackValue() Os_TimingFaultDetected() SuspendAllInterrupts() ResumeAllInterrupts() SuspendOSInterrupts() ResumeOSInterrupts()	Yes	No
コメント			

カテゴリ2 ISRによるAPIの使用

44.	カテゴリ 2 ISR が以下の API 関数を使用していないことが確認できますか？ ChainTask() ClearEvent() Os_Restart() Os_SetRestartPoint() Os_TimingFaultDetected() Schedule() StartCore() StartNonAutosarCore() StartOS() TerminateTask() WaitEvent()	Yes	No
コメント			

Disable／EnableAllInterrupts() APIの使用

45.	DisableAllInterrupts()呼び出しから EnableAllInterrupts()呼び出しまでの間にその他の API 呼び出しが行われない、ということが確認できますか？	Yes	No
コメント			

割り込みのサスペンド／レジュームを行うAPIの使用

46.	SuspendAllInterrupts()呼び出しから ResumeAllInterrupts()呼び出しまでの間、または SuspendOSInterrupts()呼び出しから ResumeOSInterrupts()呼び出しまでの間で実行される API 呼び出しは、他のサスペンド／レジューム呼び出しのみですか？	Yes	No
コメント			

割り込みのサスペンド／レジュームのネスティング

47.	Suspend／ResumeAllInterrupts()と Suspend／ResumeOSInterrupts()の呼び出しは以下の例のように正しくネストしていますか？
	
	`SuspendAllInterrupts();`
	`  SuspendAllInterrupts();`
	`  ResumeAllInterrupts();`
	`ResumeAllInterrupts();`
		Yes	No

コメント			

デフォルトのリソース

48.	RTA-OS ツールの OS Configuration > General > Optimizations において‘No RES_SCHEDULER’が選択されている場合、アプリケーションコード内で RES_SCHEDULER リソースが参照されてないことが確認できますか？ 注記: 参照されている場合は、通常はリンカによって自動検知されます。	Yes	No
コメント			

タスク／ISRがターミネートする前のリソースのアンロック

49.	タスクまたはカテゴリ 2 ISR によってロックされたリソースは、そのタスクまたは ISR がターミネートする前に必ずアンロックされますか？ 注記: GetResource()呼び出しと ReleaseResource()呼び出しは必ず一対で使用する必要があります。	Yes	No
コメント			

クリティカルセクション内でタスク切り替えが行われないこと

50.	GetResource()と ReleaseResource()の呼び出しにより定義されるクリティカルセクションの中では以下の API 関数が呼び出されない、ということが確認できますか？
	
	`ChainTask()`
	`Schedule()`
	`TerminateTask()`
	`WaitEvent()`
		Yes	No

コメント			

リソースロックのネスティング

51.	アプリケーションがリソースロックをネストさせて使用している場合、GetResource()と ReleaseResource()の呼び出しは以下の例のように正しくネストしていますか？
	
	`GetResource(res_1);`
	`  GetResource(res_2);`
	`  ReleaseResource(res_2);`
	`ReleaseResource(res_1);`
		Yes	No

コメント			

イベントのクリア

52.	アプリケーション内のイベントは、処理された後に必ずクリアされますか？ 注記: 各 SetEvent()の呼び出しごとに、対応する ClearEvent()を呼び出す必要があります。	Yes	No
コメント			

イベントのキューイングを行わない

53.	各イベントについて、クリアされる前に 2 回以上連続してセットされることがない、ということが確認できますか？ 注記: AUTOSAR OS にはイベントのキューイングメカニズムがありません。セットされたイベントが失われるのを防ぐため、イベントの処理が終了する前にそのイベントを再度セットしないように注意する必要があります。	Yes	No
コメント			

正しいチックレート

54.	すべてのソフトウェアカウンタが、コンフィギュレーションに定義された正しいレートで進められていますか？ つまり、ベースとなるタイマハードウェアは正しい周期で割り込みを提供するよう設定されていますか？	Yes	No
コメント			

コンフィギュレーションとソースコードの照合

55.	正しい API 呼び出しにより、コンフィギュレーションどおりのアラーム始動を行っていますか？ つまり、正しいオフセットと間隔で処理されるアラームを始動する SetAbsAlarm()呼び出しや SetRelAlarm()呼び出しがアプリケーションのソースコードに含まれていますか？ 注記: 自動起動されるアラームの場合は、RTA-OS の内部関数 Os_StartOS_StartAuto()内に正しい API 呼び出しが自動的に生成されるので、アプリケーションのソースコード内に上述の呼び出しを含める必要ありません。	Yes	No
コメント			

コールバックAPI

56.	満了時にコールバック関数を実行するようにアラームが設定されている場合、そのコールバック関数はすべての割り込みのサスペンドとレジュームを行うもの以外の API 関数を呼び出さないことが確認できますか？	Yes	No
コメント			

カウンタのラップアラウンド後のアラーム満了

57.	すべての SetAbsAlarm()呼び出しと SetRelAlarm()呼び出しの第 3 引数 Cycle に、アラームに対応するカウンタの OSMAXALLOWEDVALUE より小さい値を設定していますか？ 注記: OSEK は、対応するカウンタがラップアラウンドしてから N チック後にアラームが満了する、という処理を容認していません。以下の例のような API 呼び出しは禁止されています。 `SetAbsAlarm(Difficult, N, OSMAXALLOWEDVALUE + 1)`	Yes	No
コメント			

正しいチックソース

58.	すべてのスケジュールテーブルが、チックソースを提供するタスクまたはカテゴリ 2 ISR によってチックされますか？	Yes	No
コメント			

正しいチックレート

59.	すべてのスケジュールテーブルが、カウンタにより、コンフィギュレーションに定義されたレートでチックされますか？ つまり、ベースとなっているタイマハードウェアは正しい周期で割り込みを提供するように設定されていますか？	Yes	No
コメント			

16 プロジェクト情報のテンプレート

RTA-OS コード生成ツールの認定に関する各項目への回答を記入して、プロジェクト情報を完成させてください。

項目	回答
RTA-OS が使用される対象となるプロジェクト（車両プログラム）	
RTA-OS コンフィギュレーションに対する入力の種別 （例: 手書き情報、サードパーティツールで生成）	
使用される RTA-OS のバージョン （`rtaosgen --version` を実行してバージョン情報を確認してください）	
RTA-OS ポートとそのバージョン （例: PPCe200/WR、V2.0.2）	
ターゲット環境の参考資料として参照されている『RTA-OS Port Guide』の種別と、その保存場所	
RTA-OS ツールに選択されている TCL	
ECU が達成すべき ASIL	
rtaosgen.exe ツールを実行する際に使用されるコマンドラインオプション	
RTA-OS ライブラリのコンパイルに使用するコンパイラと、そのバージョン（パッチレベルを含む）	
使用される AUTOSAR 標準ヘッダファイル（Compiler.h など）のバージョン	
ECU に使用される予定のターゲットコントローラ	

項目	回答
選択されたツール認定メソッドと、その選択理由	
ツール認定結果の保存場所	
RTA-OS の機能サブセットについての記述	
その他の関連情報	