

AUTOSAR Blockset のご紹介

～ From Model to AUTOSAR software ～

© 2017 The MathWorks, Inc.

アジェンダ

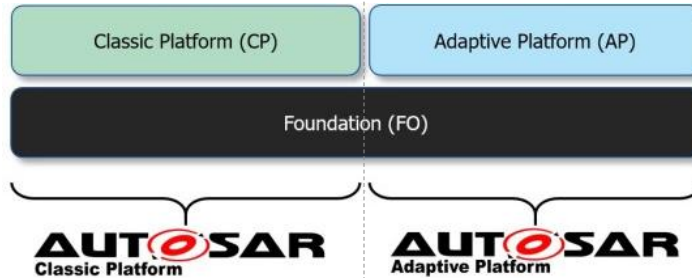
- マスワークス製品のAUTOSAR対応 について
 - AUTOSAR コード生成 (Classic)
 - AUTOSAR コード生成 (Adaptive)
 - AUTOSAR関連機能

AUTOSAR = AUTomotive Open System ARchitecture

自動車メーカーがパートナーを組んで開発・策定している標準車載ソフトウェアアーキテクチャ

Classic Platform : 2004年～
従来の制御用ECUで採用 (C言語)
固定タスクでCAN等の車内通信のみ

Adaptive Platform : 2017年～
ADAS/自動運転等のニーズを想定して策定 (C++言語)
動的スケジューリングや車外との通信も想定



マスワークスは2004年からプレミアムメンバーとして参加中

3

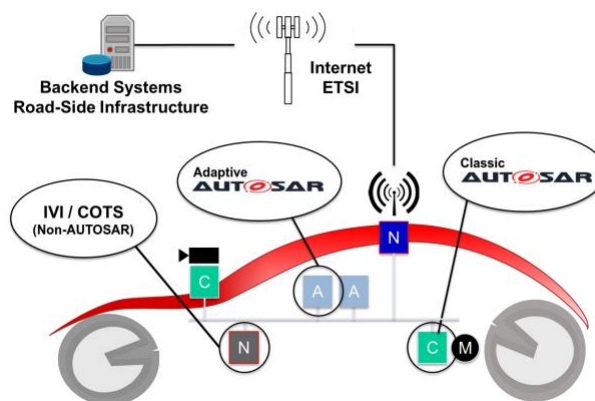
AUTOSARの適用範囲

Adaptive

- ITS, ADAS ECU
- ゲートウェイ/
ドメインコントローラー

Non- AUTOSAR

- 車載インフォテインメント
- クラスターメーター
- HUD



Classic

従来の制御系ECU

- パワトレ
- ドラトレ
- シャシー
- EPS

4

マスワークスツールのAUTOSARへの対応

R2018b以前

Embedded Coder Support Package for AUTOSAR Standard を提供しています

- Embedded Coderユーザーなら無料で利用可能です
- Classic Platformのみ対応しています
- **モデルの編集のみでもEmbedded Coderが必要です**



R2019a～

AUTOSAR Blockset を提供します

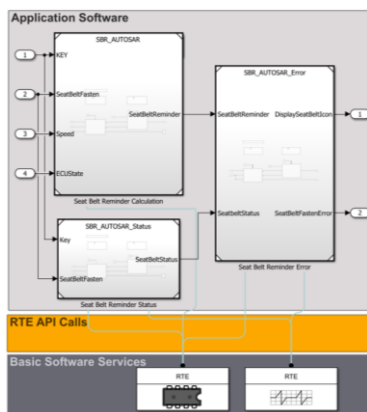
- Simulinkの有料アドオン製品です
- Classic Platformに加えてAdaptive Platformにも対応しています
- **モデルの編集はSimulinkのみで可能です。**
- コード生成にはEmbedded Coderが必要です



6

AUTOSAR Blockset 概要

- AUTOSARソフトウェアのモデリング & シミュレーション
- モデルからのAUTOSARコード & ARXMLファイルの自動生成
- AUTOSARオーサリングツールと連携した開発が可能



AUTOSAR Classic (Cコード生成)

```
void UpdateOdometerRunnable(void)
{
    uint8 rtb_TmpSignalConversionAtPulse0;
    uint16 rtb_Sum;
    rtb_TmpSignalConversionAtPulse0 =
        Rte_IrvIrReAd_UpdateOdometerRunnable_pulsedata();
    rtb_Sum = (uint16)((uint32)(uint8)(rtb_TmpSignalConversionAtPulse0 -
```

AUTOSAR Adaptive (C++コード生成)

```
boolean_T mObjectDetectionModelClass::autosar_LaneGuidance_sf_msg_pop_EvtIn(void)
{
    boolean_T isPresent;
    const ara::com::SampleContainer< ara::com::SamplePtr< const real_T > >
        *sampleContainer;
    ara::com::SamplePtr< const real_T > samples;
    if (autosar_LaneGuidance_DW.EvtIn_isValid_1) {
        isPresent = true;
    } else {
```

8

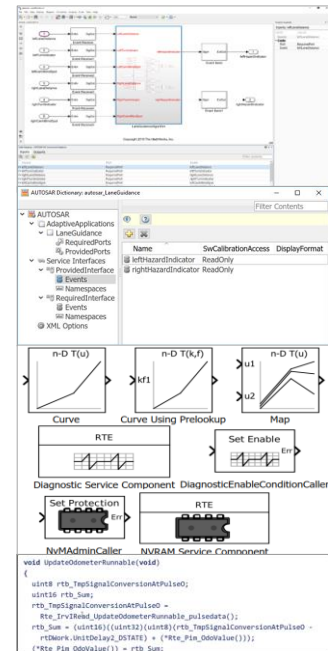
AUTOSAR Blockset 主な機能

Simulinkのみで可能

- SimulinkモデルとAUTOSARプロパティのマッピング
- ソフトウェアのプロパティ・インターフェース・データ型等を設定するディクショナリ
- BSWサービスを模擬するブロック（診断/不揮発メモリーサービス）
- AUTOSARライブラリルーチン用ブロック
- AUTOSARオーサリングツールで作成されたARXMLファイルの取り込み

Embedded Coderが必要

- AUTOSAR準拠C/C++コードおよびARXMLファイルの自動生成
- 生成コードのSIL/PIL実行



9

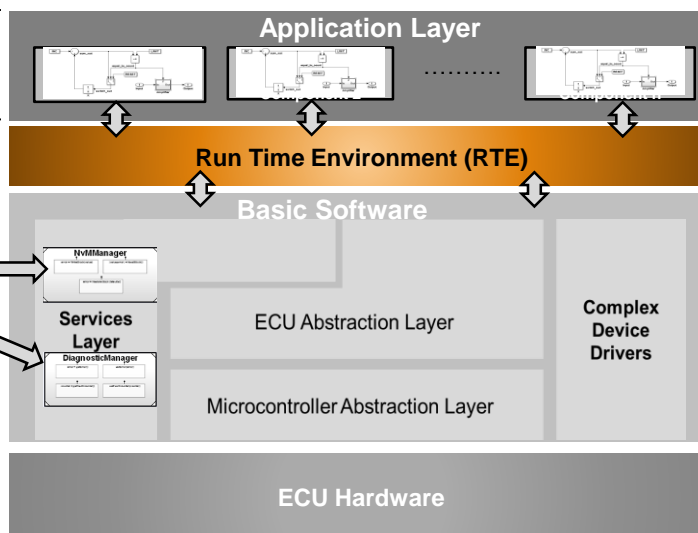
マスキュラス提供機能とAUTOSARソフト階層 (Classic) との対応

マスキュラス提供

SW-Cモデリング
& シミュレーション
& コード生成
& SIL/PIL検証



診断/不揮発メモリー
サービスブロック &
シミュレーション時
模擬



マスキュラス提供無

AUTOSAR
オーサリングツール
ソフトアーキ定義

BSW/RTE
BSWコンフィグ &
RTE生成

- Vector DaVinci
- Mentor Volcano
- EB tresos
- etc.

11

対応スキーマバージョン (R2019a時点)

Classic Platform

Schema Version Value	Schema Revisions Supported for Import	Export Schema Revision
4.3 (default)	4.3.0, 4.3.1	4.3.1
4.2	4.2.1, 4.2.2	4.2.2
4.1	4.1.1, 4.1.2, 4.1.3	4.1.3
4.0	4.0.1, 4.0.2, 4.0.3	4.0.3
3.2	3.2.1, 3.2.2	3.2.2
3.1	3.1.1, 3.1.2, 3.1.3, 3.1.4	3.1.4
3.0	3.0.1, 3.0.2, 3.0.3, 3.0.4, 3.0.5, 3.0.6	3.0.2
2.1	2.1 (XSD rev 0014, 0015, 0017, 0018)	2.1 (XSD rev 0017)

Adaptive Platform

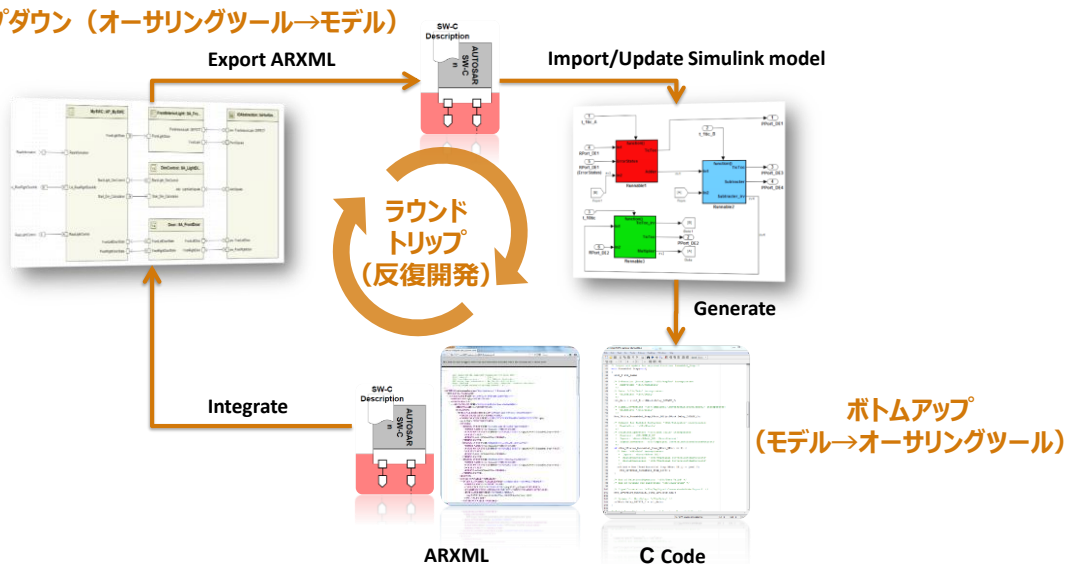
18.10

※対応スキーマバージョンはMATLABリリースにより異なります

12

モデルを用いたAUTOSAR SW-C開発ワークフロー概要

トップダウン (オーサリングツール→モデル)

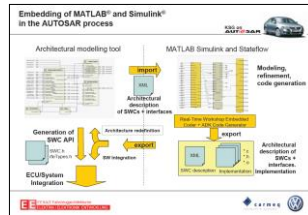


13

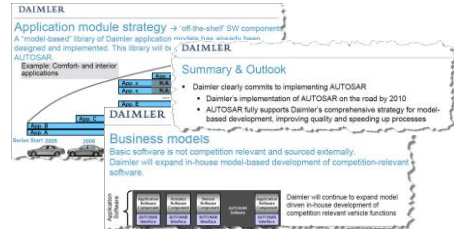
ユーザー適用事例



BMW



VWグループ

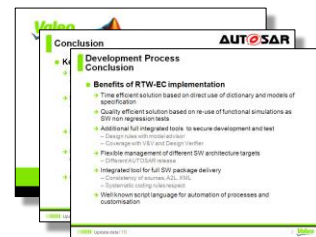


Daimler



Volvo Cars

- Set up a new Process, Method & Tool environment for future AUTOSAR based platforms
- Central data backbone (Vector eAEE)
- Model-Based Design with MATLAB, Simulink, and Embedded Coder
- ARXML import & export for system configuration data exchange



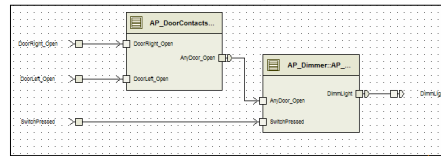
Valeo

アジェンダ

- マスワークス製品のAUTOSAR対応 について
- AUTOSAR コード生成 (Classic)
- AUTOSAR コード生成 (Adaptive)
- AUTOSAR関連機能

ボトムアップ開発

AUTOSAR Authoring Tool



⑥生成Cコード/ARXMLファイルを
オーサリングツールに取り込み

⑤SW-C Cコード/
ARXMLファイル生成

④マッピング設定の検証

①SW-Cモデル作成

②動作検証

③AUTOSARプロパティのマッピング

16

モデルに対するAUTOSARコード生成 基本設定

①システムターゲットファイルを
autosar.tlcに設定

③AUTOSAR要素とSimulink要素をマッピング

②AUTOSARコード生成オプションを設定

```
>> web(fullfile(docroot, 'ecoder/examples/autosar-code-generation.html'))
```

17

AUTOSAR プロパティ設定

- マッピングエディタからAUTOSARプロパティ情報を参照・編集できます
- 設定したプロパティは自動生成ARXMLファイルに反映されます

The screenshot displays the AUTOSAR IDE interface. The left pane shows the component tree with 'RunnableComponents' expanded. The right pane shows the 'RunnableComponents' table with columns 'Name', 'symbol', and 'canBeIn'. The 'Runnable_Init' and 'Runnable_Step' entries are highlighted. Below this, the 'DataElements' table is shown with columns 'Name', 'Invalidation', and 'SwCalibrationAcc'. The 'Input' entry is highlighted. The bottom pane shows the 'AUTOSAR Properties' section.

Name	symbol	canBeIn
Runnable_Init	Runnable_Init	false
Runnable_Step	Runnable_Step	false

Name	Invalidation	SwCalibrationAcc
Input	None	ReadOnly

The bottom pane shows the 'AUTOSAR Properties' section.

ARXMLオプション設定

XML オプションを表示して編集

パッケージ化オプション

エクスポートされた XML ファイルのパッケージ化:

パッケージパス

コンポーネント パッケージ:

データ型パッケージ:

インターフェイス パッケージ:

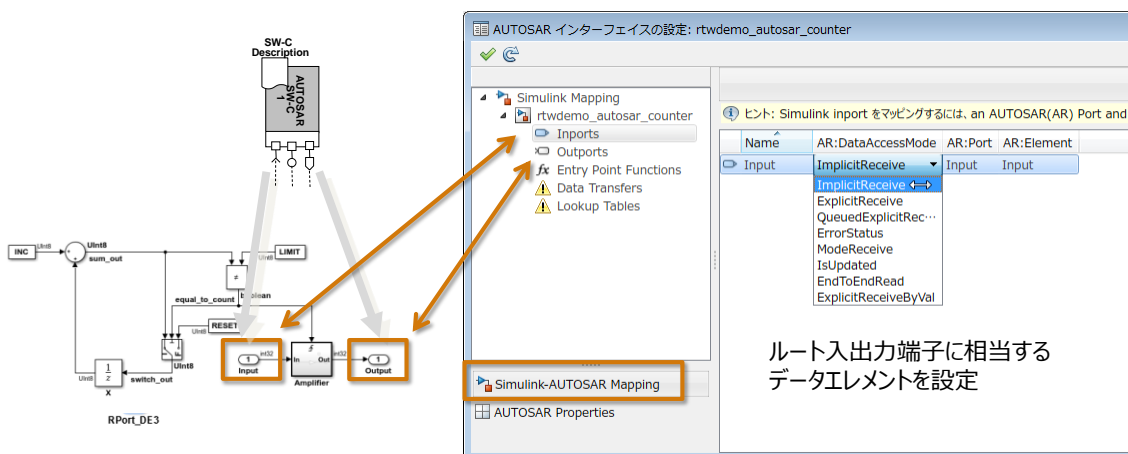
修飾名

内部動作名:

実装名:

18

AUTOSARプロパティのマッピング：入力/出力



ルート入出力端子に相当する
データエレメントを設定

19

AUTOSARプロパティのマッピング : ランナブル

The diagram shows a Simulink model on the left with blocks like INC, sum_out, LIMIT, RESET, and Amplifier. An orange arrow points from the 'LIMIT' block to the 'Simulink Mapping' section of the 'AUTOSAR インターフェイスの設定: rtwdemo_autosar_counter' window on the right.

AUTOSAR インターフェイスの設定: rtwdemo_autosar_counter

ヒント: Simulink entry point function をマッピングするには、an AUTOSAR

Name	AR:Runnable
Initialize Function	Runnable_Init
Step Function	Runnable_Step

SW-CIに含まれるランナブルを指定
(モデリングにより複数ランナブルも可能)

Simulink-AUTOSAR Mapping

AUTOSAR Properties

20

AUTOSARプロパティのマッピング : ブロックパラメータ

AUTOSAR.Parameterオブジェクトを用いてAUTOSARプロパティを設定できます

The diagram shows the same Simulink model as before. An orange arrow points from the 'LIMIT' block to the 'AUTOSAR.Parameter: LIMIT' dialog box on the right.

AUTOSAR.Parameter: LIMIT

値: 16

データ型: UInt8

次元: [1 1]

単位:

コード生成オプション

ストレータクラス: CalPrm (Custom)

HeaderFile:

ElementName:

PortName: InternalCalPrm (Custom)

InterfacePath: SystemConstant (Custom)

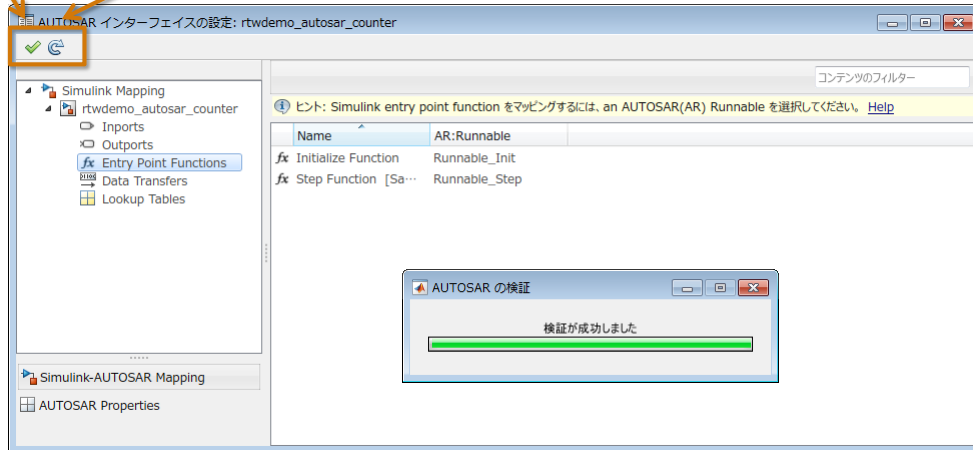
カスタム属性

- Auto
- SimulinkGlobal
- ExportedGlobal
- ImportedExtern
- Default (Custom)
- InternalCalPrm (Custom)
- CalPrm (Custom)
- SystemConstant (Custom)

21

マッピング設定の検証・同期&コード生成

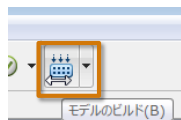
マッピング設定の検証

モデル要素のマッピングエディタ同期
(データ転送、Function-call、LUT等)

22

モデルからのコード生成

コード生成レポート

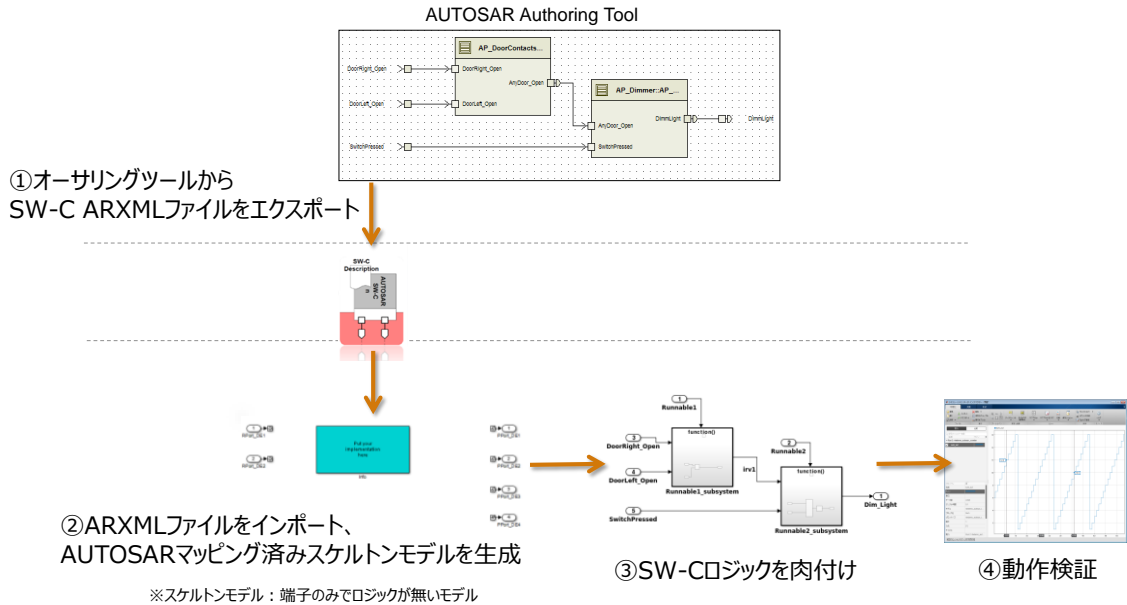


生成ファイルリスト

マッピングした
AUTOSARプロパティを
反映したコードを生成

23

トップダウン開発



24

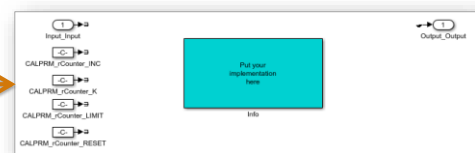
ARXMLファイルのインポート

- ARXMLプロパティインポートAPI : [arxml.importer](#)
- 上記プロパティからスケルトンモデルを作成するAPI : [createComponentAsModel](#)

```
% ARXMLファイルのインポート
importerObj = arxml.importer('rtwdemo_autosar_counter.arxml');

% スケルトンモデルの作成
createComponentAsModel(importerObj, ...
    '/rtwdemo_autosar_counter_pkg/rtwdemo_autosar_counter_swc/rtwdemo_autosar_counter');
```

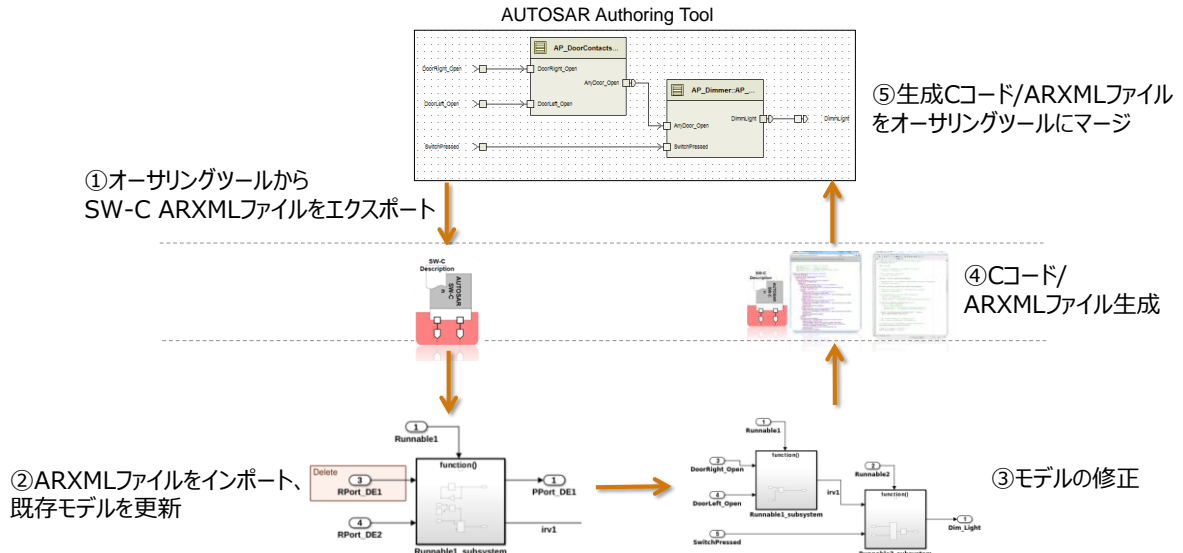
AUTOSARプロパティマッピング済み
スケルトンモデル



```
>> web(fullfile(docroot, 'ecoder/autosar/importing-an-autosar-software-component.html'))
```

25

ラウンドトリップ開発



26

ARXMLファイルインポートによる既存モデルの更新

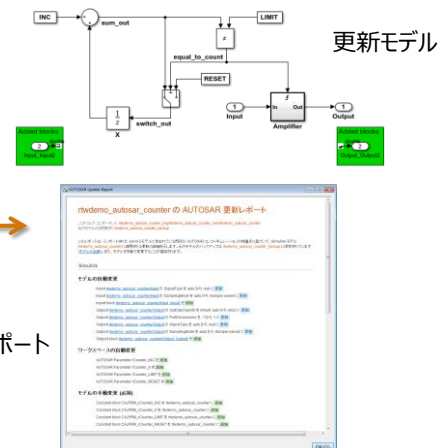
- ARXMLプロパティで既存モデルを更新するAPI : `updateModel`

```
% 更新対象のモデルを事前に開く必要があります。
open_system('rtwdemo_autosar_counter')

% ARXML ファイルのインポート
importerObj = arxml.importer('rtwdemo_autosar_counter_v2.arxml');

% 既存モデルの更新
updateModel(importerObj, 'rtwdemo_autosar_counter');
```

更新レポート



```
>> web(fullfile(docroot, 'ecoder/autosar/merge-autosar-authoring-tool-changes-into-model.html'))
```

27

アジェンダ

- マスワークス製品のAUTOSAR対応 について
- AUTOSAR コード生成 (Classic)
- AUTOSAR コード生成 (Adaptive)
- AUTOSAR関連機能

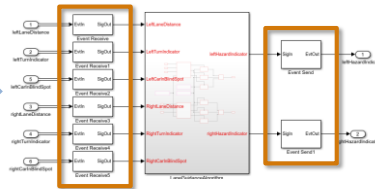
28

Adaptive C++コード生成ワークフロー

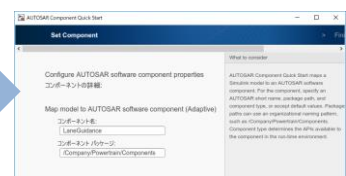
①モデルコンフィグのシステムターゲットファイルをautosar_adaptive.tlcに設定



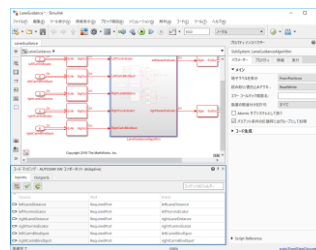
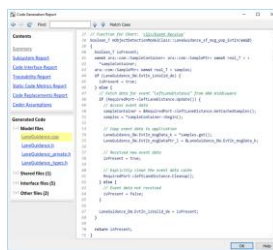
②入力にEvent Receive、出力にEvent Sendブロックを挿入



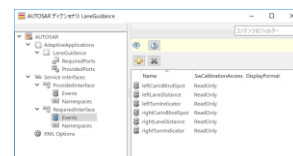
③コードパースペクティブでコンポーネント名・パッケージを設定



⑤検証後問題なければコード生成



④AUTOSARプロパティの設定 & モデル要素へのマッピング



```
>> web(fullfile(docroot, 'autosar/ug/example-create-and-configure-autosar-adaptive-software-component.html'))
```

29

Adaptive C++ 生成コードの概要

- ① イベント経由で入力データを取得
- ② 取得データを使って計算
- ③ 計算結果をイベント経由で出力

```
boolean_T ModelNameModelClass::ModelName_sf_msg_pop_EvtIn(void)
{
    // ARAミドルウェアを通じた入力端子相当データの受信
    // Update() でチェックしてGetCachedSamples()で取得
}

void ModelNameModelClass::ModelName_sf_msg_discard_EvtIn(void)
{
    // データ受信処理の終了フラグを設定
}

void ModelNameModelClass::step()
{
    // 入力データの受信
    if (ModelName_sf_msg_pop_EvtIn()) {
        ModelName_DW.SigOut = *(real32_T *)ModelName_DW.EvtIn_msgDataPtr_;
    }
    ModelName_sf_msg_discard_EvtIn();

    // モデルロジック実行
    ModelName_DW.EvtOut_msgData = 1;

    // 出力データの送信
    ModelName_sf_msg_send_EvtOut();
}

void ModelNameModelClass::initialize()
{
    // 初期化处理
}

void ModelNameModelClass::terminate()
{
    // 終了処理
    ProvidedPort->StopOfferService();
}
```

各入力に応じてメソッドが
個別に生成されます
(入力端子の数だけ生成)

- step : モデルロジック実行メソッド
- initialize : 初期化处理メソッド
- terminate : 終了処理メソッド

30

Adaptive C++ 生成コード 留意事項

- R2019a時点ではブロックパラメータは全てインライン化された数値になります。
 - Adaptive Platformで適合/測定パラメータの扱いがどうなるか決まっていないため

31

アジェンダ

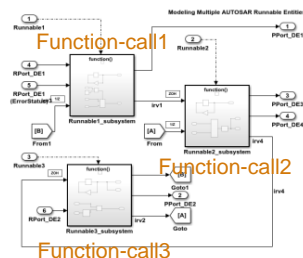
- マスワークス製品のAUTOSAR対応 について
- AUTOSAR コード生成 (Classic)
- AUTOSAR コード生成 (Adaptive)
- AUTOSAR関連機能

32

サポートされるモデリングスタイル例

- 最終的に下記がランナブルとなります
 - 周期処理
 - Function-Call
 - Simulink Function
 - Initialize / Reset / Terminate Function

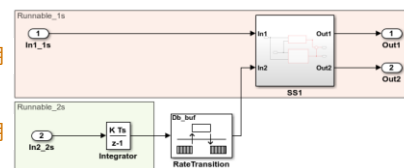
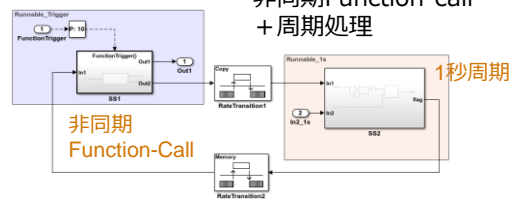
Function-callのみ



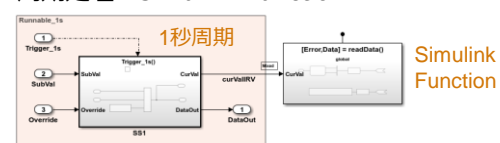
周期処理のみ

1秒周期

2秒周期

非同期Function-call
+ 周期処理

周期処理 + Simulink Function

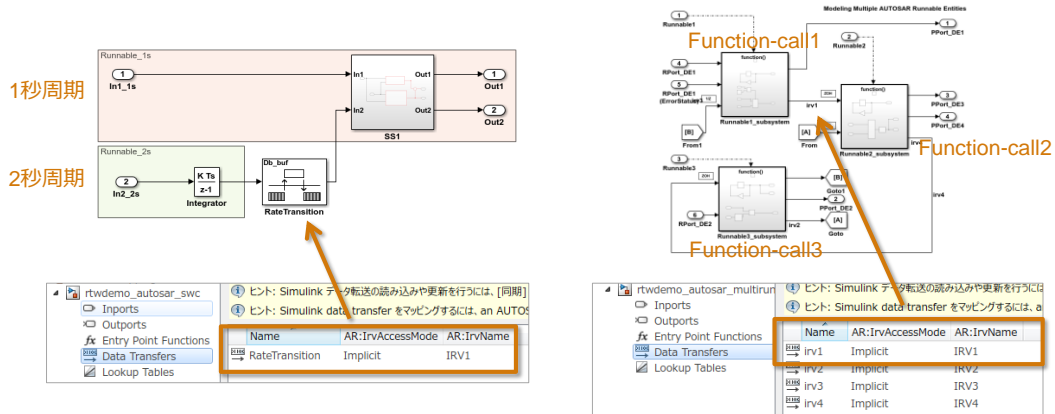


```
>> web(fullfile(docroot, 'ecoder/ug/model-for-autosar-platform.html'))
```

33

複数ランナブル間のIRV指定

- ランナブル間のIRVプロパティをマッピングエディタで指定できます
 - 異なる周期間、非同期Function-Callと周期処理間では、IRV要素としてRate Transitionブロックを挿入します
 - Function-Call間やSimulink Functionでは、IRVに相当する信号線に信号名を付けてプロパティを指定します

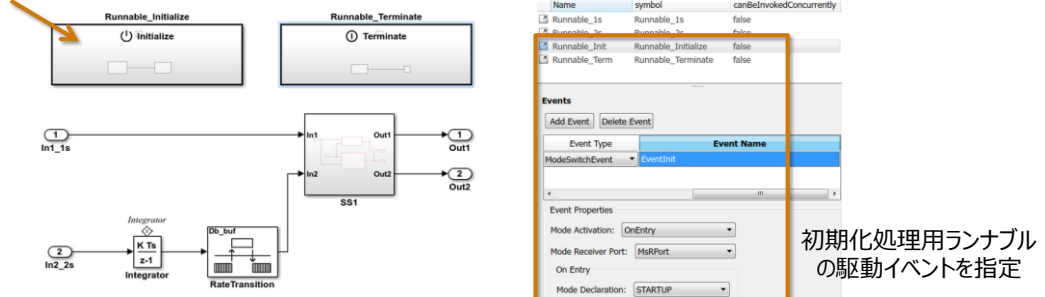


34

初期化・終了・リセット処理

- Initialize / Reset / Terminate Functionブロックを利用します
 - 状態量の初期化やデータのバックアップ処理を記述できます
 - 各Functionブロックをランナブルとして設定、イベント指定を行うことができます

積分器の状態量
初期化処理を記述



```
>> web(fullfile(docroot, 'ecoder/autosar/configure-autosar-initialization-reset-or-terminate-runnables.html'))
```

35

サーバー/クライアント

サーバー

Simulink Functionブロックで記述します



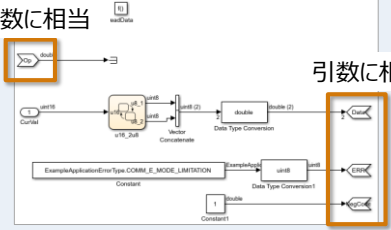
クライアント

Function Callerブロックからサーバー Simulink Functionを呼び出します



サーバー呼び出し

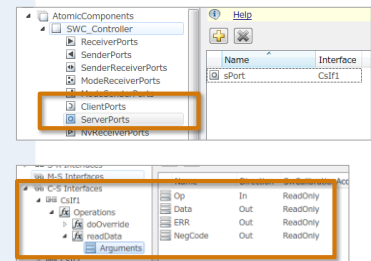
引数に相当



引数に相当



サーバー/クライアント
ポート・インターフェース設定

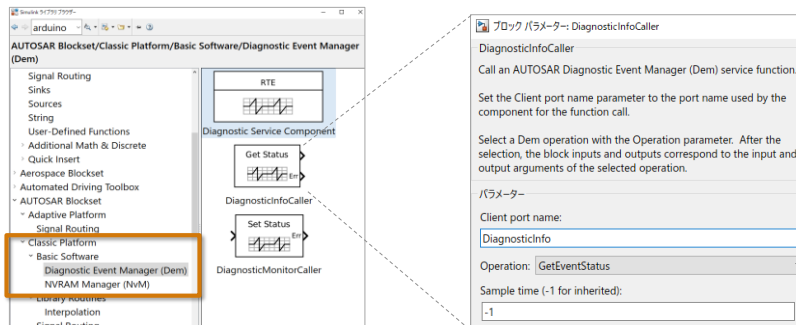


```
>> web(fullfile(docroot, 'ecoder/autosar/configure-client-server-communication.html'))
```

36

AUTOSAR BSW サービスの利用

- Basic Softwareライブラリを用いてBSWサービスを呼び出すことができます
 - 不揮発メモリ(NVM) / 診断イベント(DEM) サービスをFunction Callerブロックとして提供、モデル内に配置して利用します
 - Simulink Functionを用いて各サービスのシミュレーション時挙動を記述することもできます

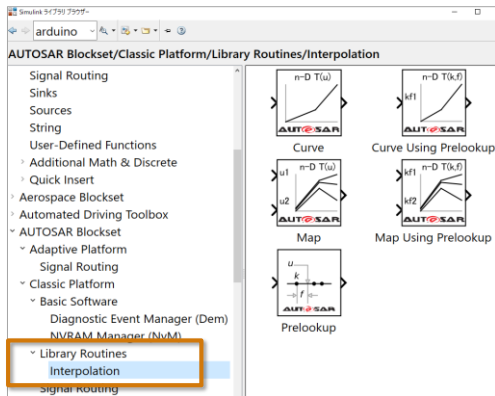


```
>> web(fullfile(docroot, 'ecoder/autosar/model-autosar-basic-software-bsw-service-calls.html'))
```

37

AUTOSAR ルックアップテーブル関数の利用

- Library Routines/Interpolationに各種ルックアップテーブルが提供されています。
- コード生成するとIFL/IFXライブラリ関数呼び出し処理が生成されます。



```
/* Model step function */
void Runnable_Step(void)
{
    If1_DPResultF32_Type rtb_Prelookup;

    /* Prelookup: '<Root>/Prelookup' incorporates:
     * Import: '<Root>/In2'
     */
    If1_DPSearch_f32(&rtb_Prelookup, Rte_IRead_Runnable_Step_In2_In2(),
        (Rte_CData_Bp_4_single())->Nx, (Rte_CData_Bp_4_single())->Bp1);

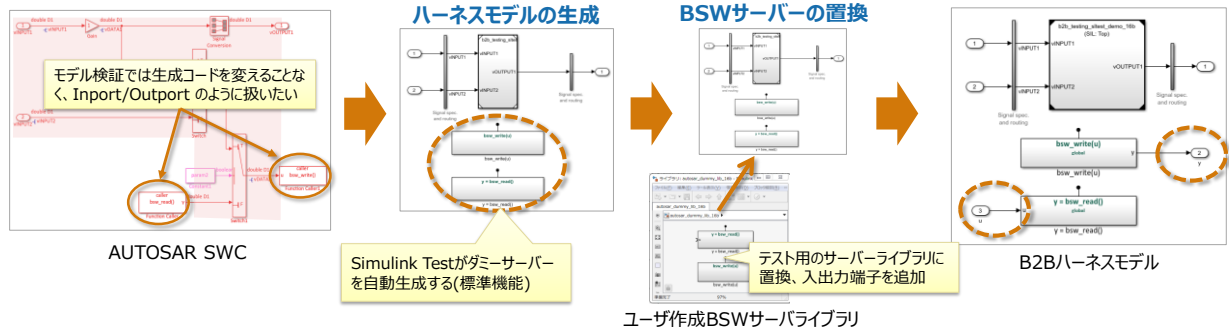
    /* Output: '<Root>/Out2' incorporates:
     * Interpolation_n-D: '<Root>/Curve Using Prelookup'
     */
    Rte_IWrite_Runnable_Step_Out2_Out2(If1_IpoCur_f32(&rtb_Prelookup,
        Rte_CData_Lcom_4_single()));
}
```

38

AUTOSAR BSWサービスを含むモデルの検証

BSWサービスをコールするSW-Cを検証する場合、Simulink Testが便利です

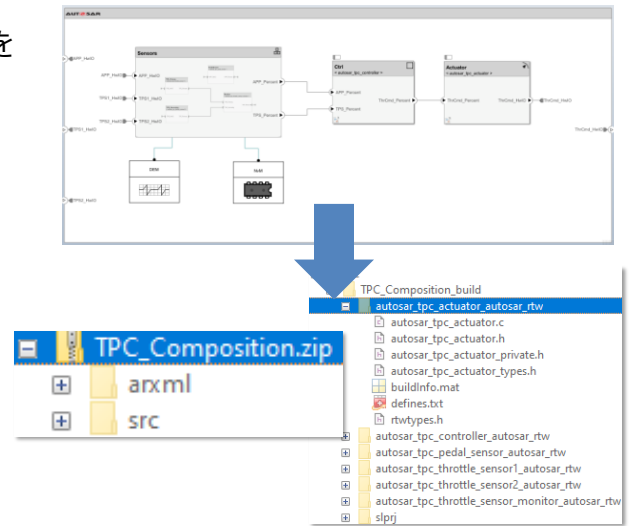
ユースケース	AUTOSAR DEM BSWライブラリを利用して、自己診断(OBD II)機能を実現している
課題	検証時にBSW経由の入出力信号を入れたいが、手動で作成する必要があり面倒
解決策	Simulink Testのテストハーネス作成機能 + BSWサーバー置換で上記課題を自動化できます



39

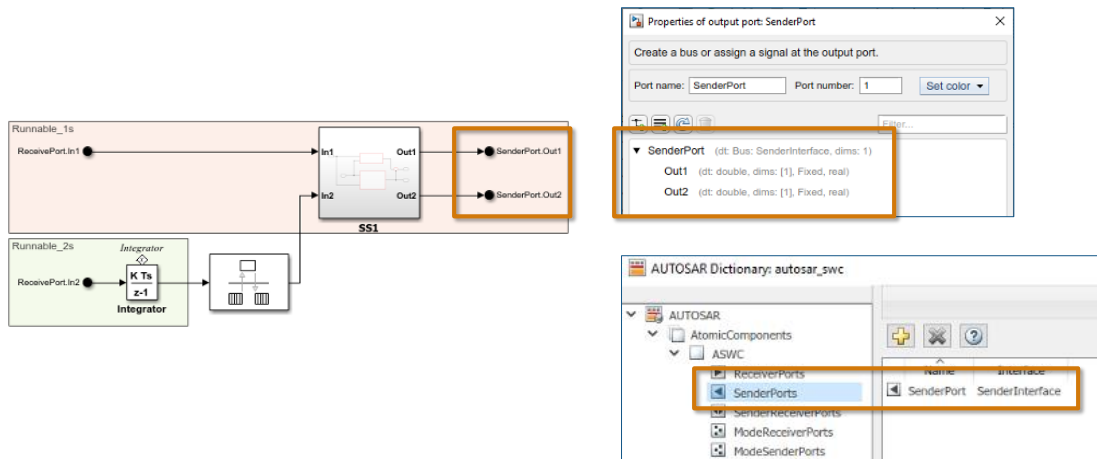
ソフトウェアアーキテクチャ図の作成

- SW-Cコンポジション・ソフトアーキテクチャを記述できます
 - System Composerライセンスが必要です
 - Classicプラットフォームのみに対応
- 各SW-CロジックをSimulinkモデルと関連付けて開発できます
- SW-Cコンポジションを含むCコードおよびARXMLコードを生成できます



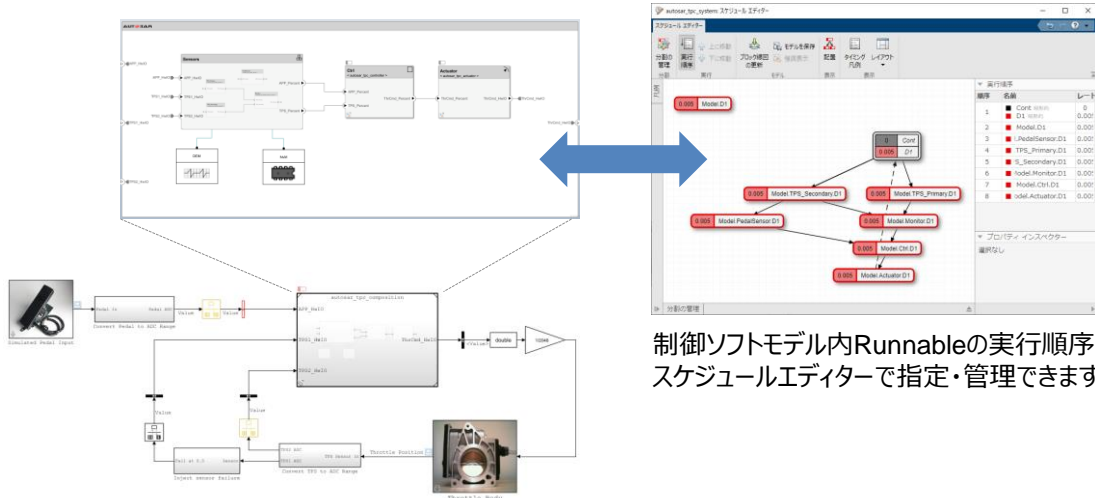
40

バス要素端子に対するAUTOSARマッピング



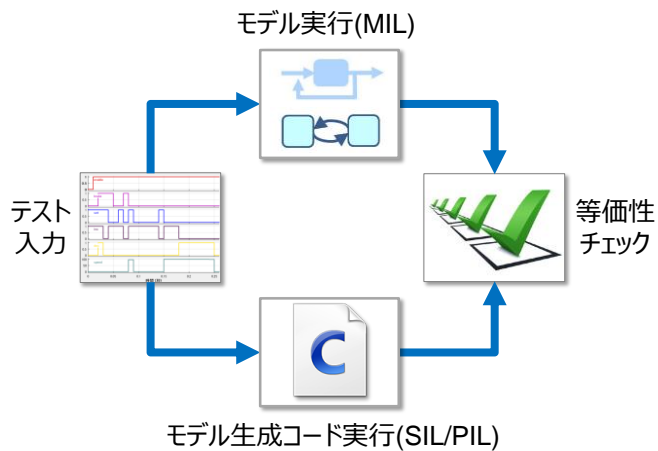
41

スケジュールエディターを用いたRunnable実行順序指定



42

AUTOSAR生成コードのSIL/PIL検証 (1/2)



Simulink Test : テスト管理・自動化
Simulink Design Verifier : テストデータ自動生成
があると便利です



SIL (Software In the Loop)

PC CPU上でコード実行



PIL (Processor In the Loop)

MCU/シミュレータ・エミュレータ上でコード実行

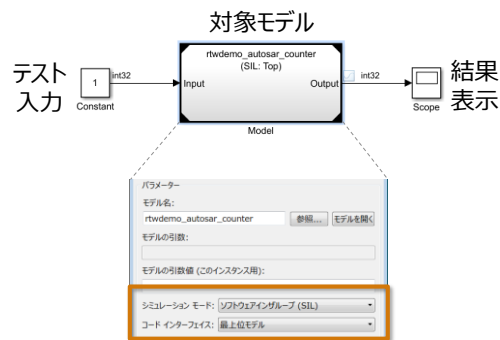
※ PIL対応しているかはMCU/IDEによって状況が異なるので要確認

- モデル＆コードの動作等価性
 - コード生成ツール設定ミス
 - コード生成ツール不具合
 - コンパイラ不具合
 - 処理系依存動作
 - メモリ消費量評価
 - 実行速度評価
- } PILのみ

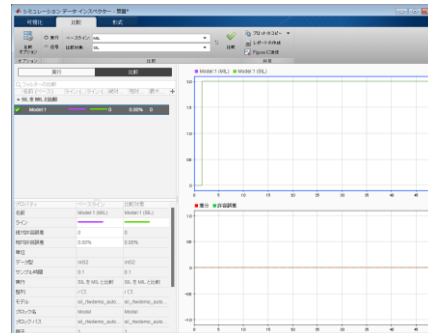
43

AUTOSAR生成コードのSIL/PIL検証 (2/2)

- モデル全体/参照モデルのシミュレーションモードとしてMIL/SIL/PILを選択できます
 - 参照モデルの場合、コードインターフェースを最上位モデルに設定する必要があります
 - RTE/VFBはスタブとして実装、SW-CロジックにフォーカスしたB2Bテストになります



シミュレーションデータインスペクターを用いたMIL/SIL比較例



```
>> web(fullfile(docroot, 'ecoder/autosar/verifying-the-autosar-code-with-sil-and-pil-simulations.html'))
```

44

膨大なAUTOSAR仕様に対応するための関連機能が提供されています ヘルプに豊富な例が紹介されています。ぜひ参考にしてください

AUTOSAR コードの生成

コンポーネントの XML 記述と C コードの AUTOSAR ランタイム環境でのエクスポート

AUTOSAR ソフトウェア コンポーネントの Simulink® モデル環境を作成および開発してから、SIL/PIL テスト用または AUTOSAR ランタイム環境 (RTE) 用に統合するためのコードを生成します。AUTOSAR のモデル化を開始するには、AUTOSAR を参照してください。

Simulink コンフィギュレーション パラメーター

Generate XML file for schema version
Maximum SHORT-NAME length
Use AUTOSAR compiler abstraction macros
Support root-level matrix I/O using one-dimensional arrays

例および操作のヒント

AUTOSAR コンポーネントの XML と C コードのエクスポート

AUTOSAR 用に設定されたモデルから、AUTOSAR コンポーネントの XML、コードと AUTOSAR 準拠の C コードを生成するには、以下を行います。

AUTOSAR ライブラリによるコード生成

モデルが AUTOSAR コード生成用に設定されて、環境により詳細に制御するコードを生成できます。

SIL および PIL による AUTOSAR C コードの生成 (ソフトウェアインザループ) および PIL (ソフトウェアコンポーネントにモデルベースの検証を実行)

サンプル ワークフロー

Embedded Code® は AUTOSAR ワークフローを示す以下の例を提供します。

例	使用方法
「AUTOSAR 準拠に準拠したコードの生成」	レポートベースの Simulink® モデルから、AUTOSAR 準拠なコードを生成し、AUTOSAR 準拠のコードを生成して、AUTOSAR XML ファイルをエクスポートします。
「複数ランタイム エンティティ用の AUTOSAR 準拠コードの生成」	複数の呼び出し元で、複数のランタイムが含まれる Simulink モデルから、AUTOSAR 準拠なコードを生成して、AUTOSAR 準拠のコードを生成して、AUTOSAR XML ファイルをエクスポートします。
「Model for AUTOSAR Platform」	レポートベースおよび参照呼び出し元ベースの方法を使用して、AUTOSAR アドミニクソフトウェア コンポーネントを Simulink でモデル化します。
AUTOSAR のプロトタイプ/マッピング 関数の例	プログラムで AUTOSAR 準拠なモデルを追加し、AUTOSAR プロトタイプ/マッピング関数を使用して、Simulink 準拠な AUTOSAR 準拠にマッピングします。
AUTOSAR クライアントサーバー連携の構成	シミュレーションおよびコード生成のために、Simulink の AUTOSAR サーバーとクライアントをモデル化します。
AUTOSAR モード変換ポートとモードスイッチイベントの構成	Simulink でモード変換ポートとモードスイッチ イベントを定義し、AUTOSAR モード スイッチ インターフェイスを設定します。

```
>> web(fullfile(docroot, 'ecoder/model-patterns-for-autosar.html'))
```

45