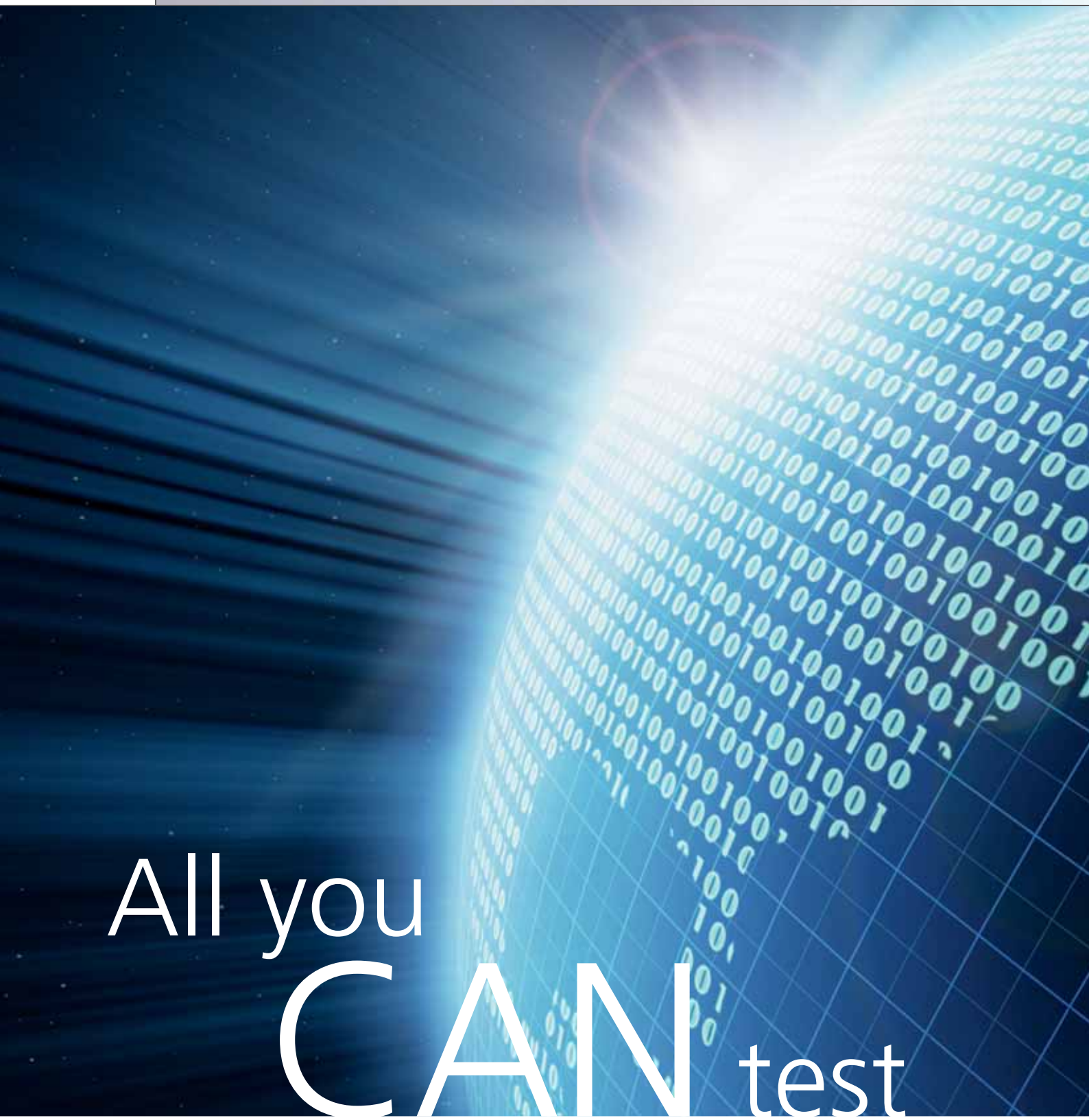

All you CAN test

CAN 環境のセットアップからリアルタイムテストまで

dSPACE の RTI CAN MultiMessage Blockset は、大規模な CAN 環境を柔軟かつ簡単に実現できる、実績のあるツールです。現行の RTI CAN MultiMessage Blockset は、CAN Navigator を介して ControlDesk に統合でき、またリアルタイムテストをサポートしているため、ユーザは開発とテストをシームレスに切り替えて実施できます。ここでは、このツールの一般的なワークフローについて説明します。

自動車開発でラピッドコントロールプロトタイピング（RCP）および HIL（Hardware-in-the-Loop）シミュレーションが適用される場合、一般的には CAN バスが使用されます。リアルタイムモデルでは、適用分野に応じて、コントローラまたは制御対象システムのいずれかが、再現されます。このモデルには CAN 通信も含まれます。CAN バスの設定は、RTI CAN MultiMessage Blockset で CAN バス信号とメッセージを定義するデータベース（DBC ファイルなど）を使用して行います。

このブロックセットでは、Simulink® 用のグラフィカルユーザインターフェース（図 1）を提供しています。ユーザは、このインターフェースを使用して必要な Rx（受信）および Tx（送信）メッセージを選択できます。ここで選択した内容が、リアルタイムモデル内でモデル化される CAN バスの基本設定になります。また、DBC バリエーション、伝送制御、および各種信号操作に関する設定を行うこともできます。これらの設定は、HIL アプリケーションで特に重要になります。HIL シミュレータの CAN 通信を後でテストし自動化する際に使用され

図 1：伝送するモデル信号を
RTI CAN MultiMessage Blockset で選択

図 2：CAN Navigator は ControlDesk で CAN を
操作するための中心的なアクセスポイント

図 3：Tx レイアウトは CAN Navigator から直接
生成することが可能

る動的介入ポイントは、これらの設定に基づいてリアルタイムアプリケーション内に生成されるためです。また、ControlDesk の CAN モニタリング機能やリアルタイムテストのための CAN サポート機能も、ブロックセット内に用意されています。

コントロールセンターとしての CAN Navigator

ControlDesk では、CAN Navigator ツリーが CAN を操作するための中心的なアクセスポイントになっています。リアルタイムモデルのためのコード生成プロセスにより、ツリーの作成に必要なすべてのデータを利用できるようになります。ユーザは ControlDesk でアプリケーションへの参照を挿入するだけで、ツリーが自動的に生成されます（図 2）。このツリーには、モデル内で定義され RTI CAN MultiMessage Blockset で設定された、一貫性のある CAN 通信設定が表示されます。

モデルに含まれるすべての CAN コントローラとこれらに割り当てられた DBC ファイルがツリーに表示され、また CAN メッセージとその信号が表示されます。実行時には、ツリーを使用して CAN コントローラの DBC 設定のバリエーションを切り替えることができます。これまでと同様、Python を使用して事前に CAN MultiMessage Blockset からレイアウトを生成できるだけでなく、必要に応じていつでもツリーからレイアウトを生成することができます（図 3）。この場合、Simulink がインストールされている必要はありません。レイアウトには、リアルタイムモデルのメッセージおよび信号の送受信設定がそのまま反映されます。このように生成されるレイアウト要素の代表的な例として、サイクルタイム設定用の入力フィールドや散発的なメッセージ送信用のボタンなどが挙げられます。

通信解析

グローバルレイアウトを生成して、複数のメッセージの伝送制御を同時に操作することもできます。CAN Navigator では、CAN ゲートウェイのオンライン設定に使用するレイアウトを作成することもできます。これはブロックセットでは作成できません。このレイアウトには、通信動作を総合的に分析するためのさまざまなビュー（図 4、図 5）やソートオプションを備えた CAN モニタリングウィンドウがあります。モニタリングは、生（RAW）データに基づいて行うか、または現在参照されている DBC を使用してシンボリックに実行することができます。

モニタリング用に選択するメッセージは、自由に定義して保存できるフィルタルールを使って絞り込むことができます。また、CAN トラフィック全体の視覚化も可能です。通過フィルタおよび阻止フィルタは、ID、ID 範囲、ECU に対して、選択的に適用できます。CAN モニタリングウィンドウで視覚化されるメッセージは、ファイル（*.csv または *.asc）に保存することも可能です。このファイルは、CAN 通信を正確なタイミングで再生（CAN 再生）する際の開始点として使用できます。たとえば、実車テスト時に記録された通信データは、必要に応じて HIL シミュレータ上のレストバスシミュレーションで簡単に再現できます。

リアルタイム条件下でのテスト

より複雑なテストシナリオを作成する場合は、Python ベースのリアルタイムテストシナリオを作成するために Real-Time Testing（RTT）が用意されています。このテストシナリオでは、テストをリアルタイムモデルと同期させて実行するので、すべてのモデル変数への読み書きアクセスを各シミュレーションクロックサイクル内で行うことができます。

リアルタイムテストで CAN バスにアクセスする必要がある場合、これらのテストは、生（RAW）データをベースにしたメッセージを読み書きするための専用ライブラリ（canmmlib）を使用して作成されます。必要なメッセージは RTI CAN MultiMessage Blockset によってモデルに追加され、リアルタイムテストまたは CAN Navigator の CAN 再生機能で利用できるようになります。この方法の利点は、CAN データベースまたはモデル構造を変更しても、リアルタイムテストを実行できるかどうかには影響を与えないことです。

次に HIL シミュレータのアナログ入力信号の 1 つをモニタリングするテストシナリオの例を取り上げます。定義されたトリガしきい値（たとえば 14.7V）を超えた場合は、値がしきい値よりも低いレベルに戻るまで、事前に定義した CAN メッセージが 50ms 間隔で周期的に送信される必要があります（図 6）。これに関連付けられるリアルタイムテストの構築については、次頁右コラムで説明しています。

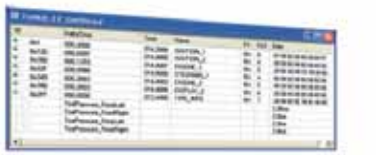

図 4：CAN メッセージ TIRE_INFO のデコードされた内容を含む静的なモニタリングビュー

図 5：連続モニタリングビュー（受信時刻でソートした場合）

ツールによる CAN 操作

RTI CAN Multi-Message Blockset によって提供される各種設定オプションにより、複数のツール間での CAN 通信設定の一貫性が保証されます。これは、対話形式による試験、CAN のモニタリングと再生、および自動化されたレストバスシミュレーションなど、ECU テストに特有なシナリオすべてに対しても同様です。このように、dSPACE では、CAN 通信を編集するための柔軟で使いやすい包括的ソリューションを提供しています。これらのソリューションは、近い将来、CAN 以外のバスシステムに対しても適用できるようになります。■

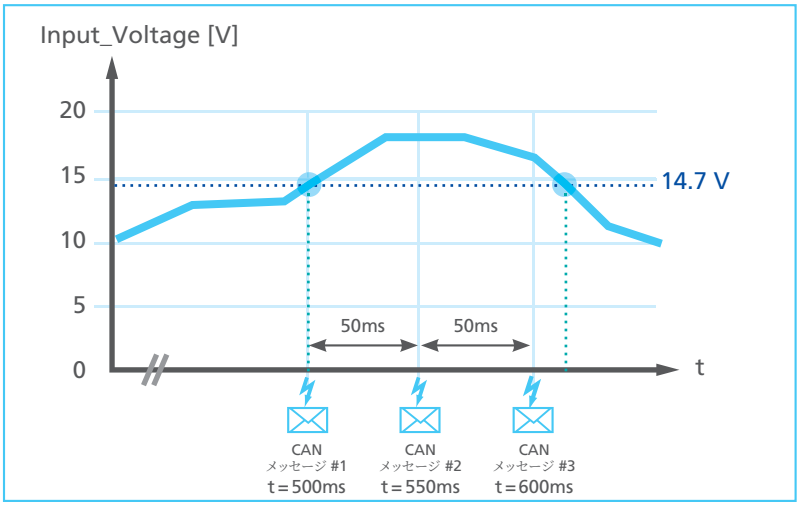

図 6：シナリオの例：モデル信号、およびそれに対するトリガされた周期的な CAN メッセージ

テストシナリオ例を用いたリアルタイムテストの構築

リアルタイムテストでは、CAN メッセージの送信データは、Python プログラミングによって自由に定義することができます。つまり、リアルタイムテストには、恒久的な動作としては使用しない各テストケース特有の CAN 動作を含めることができます。したがって、リアルタイムテストは、RTI CAN MultiMessage Blockset の静的な（モデルで定義された）CAN 設定を補完する理想的な手段となります。複数のリアルタイムテストを別々にロードしておき同時に実行する機能は、レストバスシミュレーションの実装における最大のスケーラビリティを保証します。その範囲は、図 6 の単純で反応が速いリアルタイムオブザーバから、リアルタイムハードウエア上ですべてが実行される大規模なテストシーケンスにまで及びます。

```
1. # -*- coding: ascii -*-
   from rttlib.canlib.controllers import canmmlib
   from rttlib import utilities, variable
   CurrentTime = utilities.currentTime

2. # Voltage to be observed
   Input_Voltage = variable.Variable(r'Model
   Root/Input_Voltage/Value')

3. # Initialize CANMM controller
   Controller              = None
   ControllerTRCPathName   = r"BusSystems/CAN/Chassis_M1"
   Controller              = canmmlib.GetController
                             ControllerTRCPathName)
   Channel                 = Controller.GetChannel()

4. # Initialize the response CAN message
   Message = Channel.GetRawMessage()
   Message.Format  = canmmlib.canmmbaselib.ftSTD
   Message.ID      = 0x123
   Message.DLC     = 8
   Message.TX.Data = 0x1020304050607080

5. # Wait helper function
   def WaitGen(Duration):
       EndTime = CurrentTime.Value + Duration
       while(EndTime > CurrentTime.Value):
           yield None

6. def MainGenerator():
       # Observe Input Voltage continously
       while(1):
           if(Input_Voltage.Value > 14.7):
               Message.Transmit()
               yield WaitGen(0.05)
           yield None
```

1. リアルタイムテストのライブラリをインポート（canmmlib など）
2. モニタリングする電圧の変数オブジェクトを生成
3. CAN 送信コントローラを選択
4. リアルタイムテストで送信メッセージを定義
5. 正確な時間計測に使用する補助関数（後で CAN メッセージを正確な時間間隔で送信するため）
6. リアルタイムテストシーケンス：シミュレーションのステップごとに 1 度電圧を確認し、周期的な CAN メッセージを送信