ASCET Rapid Prototyping V5.5

ユーザーズガイド

著作権について

Document EC012401 R5.5.1 JP

目次

1 はじめに

ECU ソフトウェアをリアルタイム環境でテストを行うには、リアルタイム処理を実行できる実験用ハードウェアシステムが必要です。このハードウェア（ES1000 システム）のプロセッサノードとしての役割を果たすシミュレーションボードは、「実験ターゲット」または「E-Target」（Experimental Target）と呼ばれ、本製品 ASCET-RP V5.5 （**ASCET R**apid **P**rototyping V5.5）を使用して ASCET の実験環境に統合します。そしてさらに各種 I/O ボードを統合することにより、強力な開発システムを構築することができます。

ASCET-RP の製品パッケージには、コンパイラコールやリンカコールの統合機能や実験ターゲットのための ERCOS^{EK} オペレーティングシステムカーネル、といった ASCET 開発環境の拡張機能に加え、コンパイラツールセットのライセンスも含まれています。

ASCET-RP には、ASCET 基本システムの基本機能も含まれています。モデルの作成や編集を行うには、別製品「ASCET-MD」が必要です。

1.1 コンポーネント

ASCET-RP V5.5 には、以下の機能を実現するコンポーネントが含まれています。

- ES1130 または ES1135[1] を実験環境に統合
- I/O ボード、または ETK バイパスや CAN バイパス機能を、実験ターゲット（ES1130 および ES1135）に統合 （RTIO パッケージ）
- GNU コンパイラ
- 各種ドキュメントとサンプルファイル

1.2 インストール

ASCET 基本システムと ASCET-RP は、同じ CD-ROM に納められています。インストールの際は、ASCET 基本システムをインストールしてから ASCET-RP V5.5 をインストールし、さらに所定のライセンスファイルを取得する必要があります。

最初に `ASCET.exe` というインストールプログラムを実行し、続いて `ASCET-RP.exe` を実行してください。

ASCET-RP V5.5 のインストールについての詳細は、リリースノートを参照してください。

ライセンスについては『ASCET 入門ガイド』に説明されています。

1. 本マニュアル内においては、任意のシステムコントローラを表わす場合は「ES113x」という名称が使用されています。

サンプルファイル

ASCET-RP V5.5 をインストールすると、ASCET からエクスポートされたサンプルデータベース（`RTIOTutorial.exp` および `INTECRIO_Tutorial.exp` というエクスポートファイル）が、ASCET ディレクトリ内の `EXPORT` ディレクトリに格納されます。

1.3 本書の構成

この『ASCET-RP V5.5 ユーザーズガイド』は、以下の 3 つの内容で構成されています。

- 概要
- リアルタイム I/O パッケージの詳細（バイパスインターフェースの機能説明を含む）
- チュートリアル

概要の部分は ASCET-RP V5.5 のすべてのユーザーの方を対象としています。ここでは ASCET-RP V5.5 の構成とインストール、および一般的な使用方法に関して説明されています。

次の部分に、ASCET-RP V5.5 に含まれる RTIO（リアルタイム I/O）パッケージの機能や操作方法についての詳しい情報が記載されています。

チュートリアルには、いくつかのボードの設定方法と INTECRIO での実験方法について説明されています。

1.4 本書の記述について

1.4.1 記述形式

ユーザーが実行するすべてのアクションは、以下のような“Use-Case”形式で記述されています。

目標の定義：

- 手順 1

  手順 1 についての説明
- 手順 2
- 手順 3

上記のように、操作を行う目標がタイトルとして最初に簡潔に定義され（例：「新しいコンポーネントを作成する」、「エレメントの名前を変更する」）、その下に、その目標を実現するために必要な操作手順が列挙され、必要に応じて ASCET のウィンドウやダイアログボックスのスクリーンショットが添付されています。

1.4.2 表記上の規約

本書は以下の規則に従って表記されています。

表記例	説明
File → Exit を選択して、...	メニューコマンドは、**青の太字**で表記します。
OK をクリックして、...	ユーザーインターフェース上のボタン名は、**青の太字**で表記します。
<Ctrl> を押して、...	キーボードの各キーは、**< >** 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、入力フィールド等のタイトルは、“ ”で囲んで表記します。
`setup.exe` ファイルを選択します。	リストボックス、プログラムコード、ファイル名、パス名等のテキスト文字列は、`Courier` フォントで表記します。
論理型のデータから算術型のデータへの変換はできません。	注意すべき個所、または新出の用語は**太字**、または「 」で囲んで表記されます。
OSEK グループ（http://www.osek-vdx.org/ を参照してください）はさまざまな標準規格を策定しています。	インターネットへのリンクは、青い下線で表記されています。

特に重要な注意事項は、以下のように表記されています。

注記

ユーザー向けの重要な注意事項

また PDF 文書において、索引、および他の部分を参照する個所（例：「xxxx を参照してください」の「xxxx」の部分）については、その参照先へのリンクが設けられているので、必要な参照個所を素早く見つけることができます。

2 実験ターゲットの設定

ASCET-RP V5.5（**ASCET R**apid **P**rototyping V5.5）には、トランスピュータまたは PowerPC を搭載した各ターゲット用の実行ファイルを作成するために必要なコンパイラ／リンカツールと、ハードウェアを統合するための ASCET 開発環境の拡張機能が含まれています。ASCET のプロジェクトエディタにおいてターゲット（Target オプション）を選択するだけで、そのターゲット用のすべての機能が ASCET の環境に統合されます。

コンパイラとリンカのコンフィギュレーション、および実際のターゲットハードウェア用インターフェースは、ASCET ソフトウェアから直接定義するのではなく、ETAS ネットワークマネージャ、または一連の `*.ini` ファイルで定義します。2.1 項では ASCET-RP ソフトウェア内で行うハードウェアオプション設定について説明し、2.2 項で ETAS ネットワークマネージャを利用したハードウェア接続の設定方法について説明します。また 2.3 項では `target.ini` ファイルを変更してホストインターフェースの設定を行う方法やイーサネットインターフェースの設定方法について説明します。

2.4 項ではコンパイラの選択と設定について説明します。

Power-PC を使用した実験ターゲット用ディレクトリの構造

ASCET-RP V5.5 をインストールすると、ASCET ディレクトリの下に 2 つのターゲットディレクトリ（PowerPC ベースの各実験ターゲット用のサブディレクトリ）が作成され、各ターゲットに固有な情報や設定、およびライブラリファイルが格納されます。

ASCET の サブディレクトリ	**実験システム（VME システム）**	**実験ターゲット（シミュレーションボード）**
`..¥Target¥ES1130`	ES1000.2/ES1000.3[a]	ES1130
`..¥Target¥ES1135`	ES1000.2/ES1000.3	ES1135
`..¥Target¥Prototyping`	（INTECRIO で設定）	

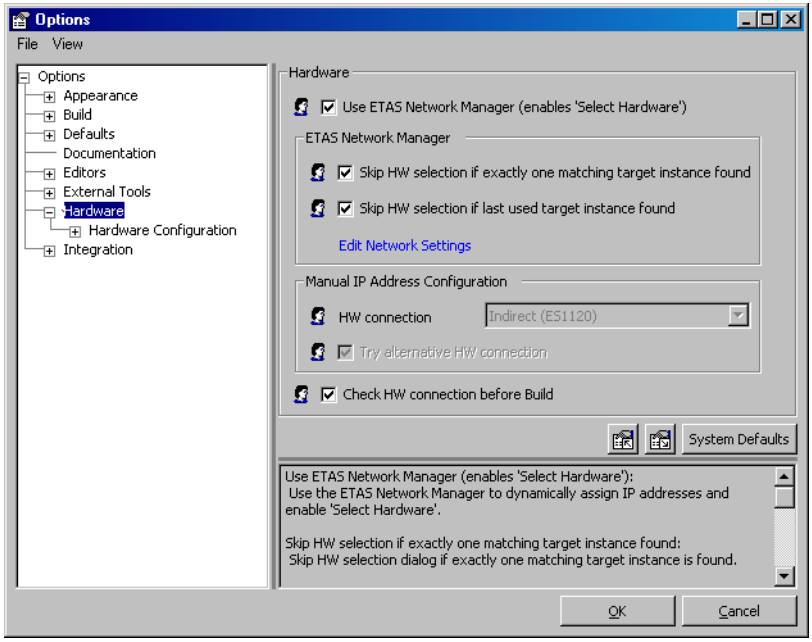
a: 本マニュアル内において「実験システム」は、特定の機種に限定した説明部分を除き、通常は「ES1000」または「ES1000.x」という名称で記載されています。

両方のシミュレーションボード（ES1130 および ES1135）で共通に使用されるファイルは、`..¥target¥ES113x` ディレクトリに格納されています。

実験ターゲットを変更する際、ASCET の “Options” ウィンドウのシステムルートパス設定を変更する必要はありません。

2.1 ハードウェアに関するオプション設定

ASCET-RP をインストールすると、ASCET の“Options”ウィンドウに“Hardware”ノードが追加され、ここでハードウェアインターフェースの設定を簡単に行うことができます。以下にこのタブで設定できるオプションについて説明します。通常は、最も頻繁に使用するハードウェア構成を設定しておいてください。



- **Check HW connection before Build**

  実験開始時（**Open Experiment**）、このオプションがオン（デフォルト）になっていると、ビルド処理の前後にハードウェア検索が行われ、オフになっているとビルド処理の前にのみ行われます。検索によって適切なハードウェアが検知されなかった場合、エラーメッセージが出力されます。

  このオプションがオンになっていれば、未接続のハードウェアをビルド処理実行中に接続できるので、時間の無駄がなくなります。

  オプションがオフになっていると、未接続のハードウェアがあっても、ビルド処理時にエラーメッセージが出力されません。

- **Use ETAS Network Manager (enables 'Select Hardware')**

  ETAS ネットワークマネージャ（第 12 章を参照してください）を使用するかどうかを指定するためのオプションです。デフォルトではオンになっています。
  このオプションがオンになっていると、**Select Hardware** ボタン、およびメニューコマンド **Extras → Select Hardware** が使用可能となります。

注記

以下の 2 つのオプションとリンクは、ETAS ネットワークマネージャの使用が有効になっている場合にのみ意味を持ちます。

- **Skip HW selection if exactly one matching target instance found**

  このオプションがオン（デフォルト）になっていると、実験開始時のハードウェア検索によって、プロジェクトで定義されたものと一致するハードウェアが 1 台のみ検知された場合、"Experimental Target Hardware Selection" ダイアログボックスが開きません。
  このオプションがオフになっていると、次のオプション（**Skip HW selection if last used target instance found**）の設定によって、実験開始時に "Experimental Target Hardware Selection" ダイアログボックスが開くかどうかが決まります。このダイアログボックスには、PC に接続されているすべての実験ハードウェアが表示され、使用するものを選択することができます。
- **Skip HW selection if last used target instance found**

  このオプションがオン（デフォルト）になっていると、実験開始時のハードウェア検索によって、前回プロジェクトで使用されたハードウェアのみが検知された場合、"Experimental Target Hardware Selection" ダイアログボックスが開きません。
  このオプションがオフになっていると、前のオプション（**Skip HW selection if exactly one matching target instance found**）の設定によって、実験開始時に "Experimental Target Hardware Selection" ダイアログボックスが開くかどうかが決まります。
- **Edit Network Settings**

  このリンクをクリックすると ETAS ネットワークマネージャ（第 12 章を参照してください）が開きます。

注記

以下の 2 つのオプションは、ETAS ネットワークマネージャの使用が**無効**になっている場合にのみ意味を持ちます。

- **HW connection**

  このコンボボックスで、ES1000 システム内の ES1120 コントローラボードと PC を接続するか（`Indirect (ES1120)`、デフォルト）。それとも ES113x シミュレーションボードと PC を直接接続するか（`Direct (ES113x)`）を選択します。

- **Try alternative HW connection**

  ハードウェア検査時、このオプションがオン（デフォルト）になっていると、“HW connection”コンボボックスで選択されたもの以外のデバイスも検索され、オプションがオフになっていると、選択されたデバイスのみが検索されます。

2.2 ETAS ネットワークマネージャを用いたハードウェア接続設定

ETAS ネットワークマネージャ（ETAS Network Manager）を使用すると、さまざまなハードウェア接続環境を詳細に設定できます。

- 1 つのネットワークアダプタを、ETAS ハードウェアと社内ネットワークの両方で共有できます。
- 個々のネットワークアドレスを指定できます。
- ETAS ハードウェアネットワーク内から、使用するシミュレーションボード（ES113x）を選択できます。

ETAS ネットワークマネージャを使用してハードウェア接続を設定する方法は、本書の第 12 章で説明されています。

ETAS ネットワークマネージャの使用を有効にする：

- ASCET コンポーネントマネージャから、**Tools → Options** を選択します。

  “Options”ダイアログボックスが開きます。
- “Hardware”ノードを開きます。
- **Use ETAS Network Manager (enables ‘Select Hardware’)** オプションをオンにします。

  このオプションのみがオンになっていると、実験を開始するたびに“Experimental Target Hardware Selection”ダイアログボックスが開きます。
- **Skip HW selection if ...** オプションをオンにすると、所定の条件に該当するハードウェアのみが接続されている場合、ハードウェア選択ダイアログボックスが開きません。
- ビルド処理の実行前にもハードウェア検索が行われるようにするには、**Check HW connection before Build** オプションをオンにします。

“Experimental Target Hardware Selection”ダイアログボックスは、マニュアル操作で任意に開くこともできます。

マニュアル操作でハードウェア選択ダイアログボックスを開く：

注記

この操作で使用する **Select Hardware** ボタン、および **Extras → Select Hardware** メニューコマンドは、**Use ETAS Network Manager (enable 'Select Hardware')** オプションがオンに設定されている場合のみ、使用可能です。

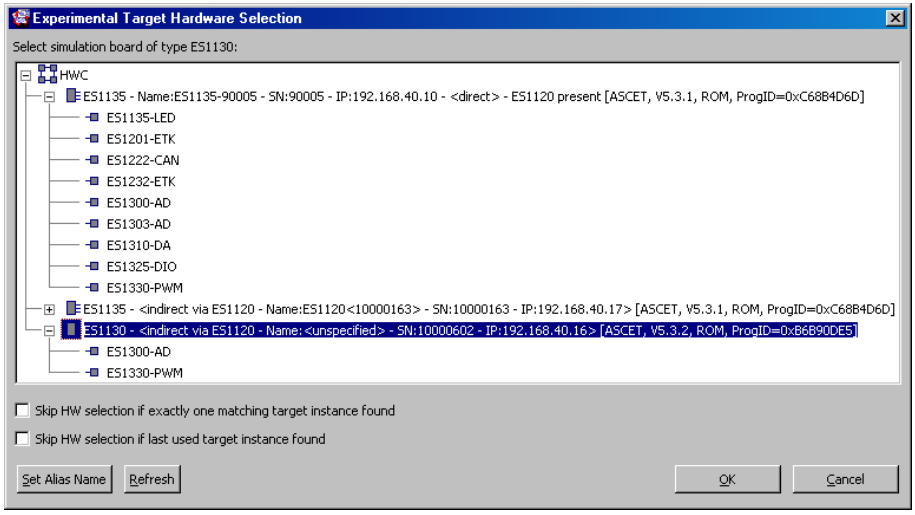
- **Extras → Select Hardware** を選択します。

または

- **Select Hardware** ボタンをクリックします。

  ハードウェア選択ダイアログボックスが開きます。

2.2.1 ハードウェア選択ダイアログボックス



このダイアログボックス（“Experimental Target Hardware Selection”ダイアログボックス）には以下のエレメントが含まれています。

- “Select simuation board of type *<type>*”[1] フィールド

  このフィールドには、メインアイテム `HWC`（シンボル  ）の下に、PC に接続されているすべてのシミュレーションコントローラ（ES113x、シンボル  ）が一覧表示されます。シミュレーションコントローラのラベルには、ES113x の接続方法（`direct` または `indirect`）などの情報が含まれます（20 ページ参照）。接続されている各種ボード（シンボル  ）は、シミュレーションボードの下に表示されます。

  ここで、プロジェクトのコード生成オプションで選択されているシミュレーションボードを選択してください。

- **Skip HW selection if exactly one matching target instance found** オプション、および **Skip HW selection if last used target instance found** オプション

  これらのオプションは、ASCET オプションダイアログボックスの“Hardware”ノードのオプション（2.1 項参照）と同じです。ここで設定された内容は、“Hardware”ノードに反映されます。

- **Set Alias Name** ボタン

  このボタンで、ES113x または ES1120 に任意の名前を割り当てることができます。

- **Refresh** ボタン

  このボタンをクリックすると、“Select simulation board of type *<type>*”フィールドの内容が更新されます。新たに接続したり電源をオンにしたハードウェアが表示され、切り離したり電源をオフにしたハードウェアは表示されなくなります。

- **OK** ボタンと **Cancel** ボタン

  選択を確定するに **OK** ボタンをクリックし、設定内容をキャンセルしてダイアログボックスを閉じるには **Cancel** ボタンをクリックします。

直接 PC に接続されているシミュレーションボードについては、“Selection simulation board of type *<type>*”ダイアログボックス内のラベルは以下のようになります。

```
ES113x - Name:<alias> - SN:<serial number> -
   IP:<IP address> - <direct> - ES1120 present
   [<SW>, <syslib version>, <boot mode>,
   ProgID=<ID>]
```

- `ES113x` はシミュレーションボードのラベルです。
- `Name:<alias>` は、シミュレーションボードに任意に割り当てられた名前です。
  名前が割り当てられていない場合、この部分は表示されません。
- `SN:<serial number>` は ES113x のシリアル番号です。

[1] *<type>* の内容は、プロジェクトで選択されているターゲットです。

- IP:<*IP address*> は ES113x の IP アドレスです。
- <direct> は、ES113x が直接 PC に接続されていることを示します。
- ES1120 present は、ES1000 システム内に、未接続の ES1120 が組み込まれていることを示します。
  ES1000 に ES1120 が 1 枚も組み込まれていない場合、この部分は表示されません。
- <*SW*> は、プログラムを ES1000 にロードする際に使用したソフトウェア（ASCET、INTECRIO など）です。
- <*syslib version*> は、使用されているハードウェアシステムライブラリのバージョンです。
- <*boot mode*> は、プロジェクトが ES1000 の電源投入時に FLASH メモリから起動されたか（ROM）、またはパワーオン後にダウンロードされたものであるか（RAM）を示します。
- ProgID=<*ID*> は、ソフトウェア <*SW*> によってプロジェクトに割り当てられた識別子（<*ID*>）を示します。

ES1120 などを経由して間接的に PC に接続されているシミュレーションボードについては、ラベルは以下のように表示されます。

ES113*x* - <indirect via ES1120 - Name:<*alias*> -
SN:<*serial number*> - IP:<*IP address*>>
[<*SW*>, <*syslib version*>, <boot mode>, ProgID=<*ID*>]

- ES113*x* *はシミュレーション*ボードのラベルです。
- <indirect via ES1120 ...> はES113xが間接的にPCに接続されていることを示します。
- Name:<*alias*> は、ES1120 に任意に割り当てられた名前です。
- SN:<*serial number*> は ES1120 のシリアル番号です。
- IP:<*IP address*> は ES1120 の IP アドレスです。
- <*SW*>、<*syslib version*>、<*boot mode*> 、ProgID=<*ID*> は、直接接続の場合と同じです。

2.3 ETAS ネットワークマネージャを用いないインターフェース設定

旧バージョンとの互換性のため、ETAS ネットワークマネージャを使用せず、代わりに各ターゲット用の target.ini ファイルで ASCET 用のイーサネットインターフェースを設定することもできます。このファイルは、..¥target¥ES1130 または ..¥target¥ES1135 というディレクトリに格納されています。

ES1000 の接続インターフェースを指定する：

- ASCET コンポーネントマネージャから、**Tools → Options** を選択します。
  “Options” ダイアログボックスが開きます。

- “Hardware”ノードを開きます。
- **Use ETAS Network Manager (enables ‘Select Hardware’)** オプションをオフにします。
- “HW connection”コンボボックスで、使用するES1000 の接続タイプを選択します。
- “HW connection”コンボボックスで選択されたデバイス以外のデバイスも検索されるようにするには、**Try alternative HW connectionse ETAS Network** オプションをオンにします。

  このオプションがオフになっていると、選択されているデバイス以外は検索されません。
- ビルド処理の前後にハードウェア検索が行われるようにするには、**Check HW connection before Build** オプションをオンにします。

  このオプションがオフになっていると、ハードウェア検索はビルド処理の後にしか実行されません。
- **OK** をクリックして設定を確定します。

選択内容に応じて、各ターゲット用サブディレクトリ内の `target.ini` ファイルから適切な IP アドレス変数が読み込まれ、ASCET の実験環境においてそのアドレスが使用されます。

- コントローラボード ES1120 と ASCET のホスト PC 間の通信には、以下の変数が使用されます。
  - **ES1130**

    ```
    IndirectIpAddress=192.168.40.10
                    ;Default IP-Address for ES1120.x
    ```
  - **ES1135**

    ```
    IndirectIpAddress=192.168.40.10
                    ;Default IP-Address for ES1120.x
    ```
- シミュレーションボード ES113x と ASCET のホスト PC 間の通信には、以下の変数が使用されます。
  - **ES1130**

    ```
    DirectIpAddress=192.168.40.11
                    ;Default IP-Address for ES1130.x
    ```
  - **ES1135**

    ```
    DirectIpAddress=192.168.40.15
                    ;Default IP-Address for ES1135.1
    ```

ES1000.x システムの接続に使用されるネットワークインターフェースを PC にインストールして設定するための情報は、以下のドキュメントに記載されています。

- ES1000 Ethernet InstallationGuide.pdf

このドキュメントは、ASCET のインストール時に ..¥ETAS¥ETASManuals¥ASCET V5.2¥ES1000 というディレクトリに保存されます。

2.4 コンパイラの選択

ASCET-RP には、ES1130 および ES1135 ターゲット用の GNU クロスコンパイラ（ユーザインターフェース上では GNU-C V3.4.4 (Power PC) と表示されます）と、同コンパイラの旧バージョンである GNU-C V2.95.3 (Power PC) が含まれています。また、ES1130 には Diab Data コンパイラ（ASCET-RP には含まれていません）も使用できます。

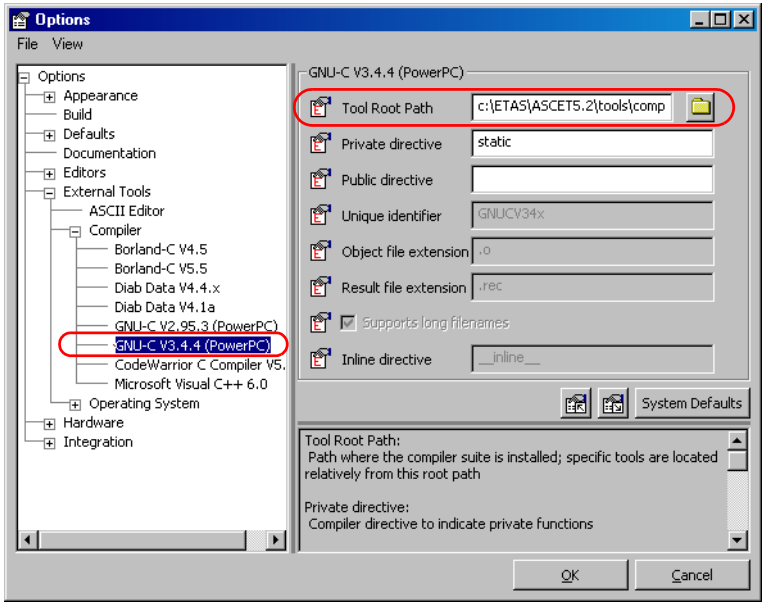
ターゲット	使用できるコンパイラ
ES1130	GNU Cross Compiler、Diab Data V4.1a
ES1135	GNU Cross Compiler

表 2-1　各ターゲット用に使用できるコンパイラ

2.4.1 任意のコンパイラを使用する

ASCET-RP V5.5 の製品 パッケージには、標準ツールとして MS-DOS 版の GNU クロスコンパイラが含まれています。その他のコンパイラを使用する場合は、以下の点に注意してください。

1. ASCET の“Options”ウィンドウで、GNU-C V3.4.4 (Power PC) のコード生成パスを、使用するコンパイラのパスに合わせて変更してください。



2. 使用するターゲット（ES1130 または ES1135）の comptool.ini ファイル内の RelativeToolPath 変数（コンパイラのパスを示す変数）の値を、RelativeToolPath=¥powerpc-eabi¥bin から、たとえば RelativeToolPath=¥win32¥bin に変更します。

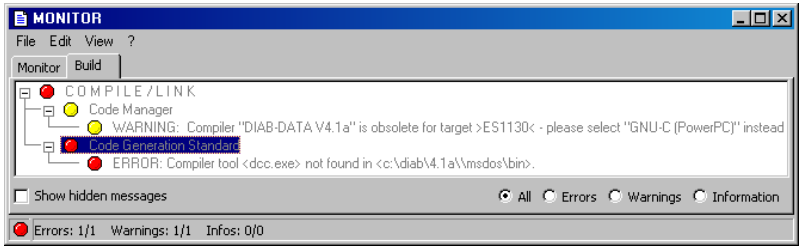
注記

パスとシステム変数は、必ず ASCET で使用するコンパイラバージョンのものを設定してください。

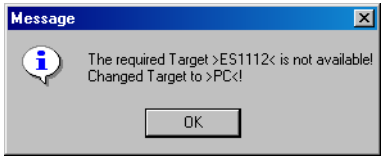
2.4.2 GNU クロスコンパイラへの変更

ASCET の“Options”ウィンドウの“External Tools/Compiler”ノードのサブノードには、Diab Data コンパイラのデフォルトパスが設定されていますが、これは、実際にコンパイラがインストールされていない場合でも、デフォルト値として設定されます。これと同様に、コンパイラがインストールされていなくても、各プロジェクトのコード生成オプション内で Diab Data コンパイラを選択することが可能です。

Diab Data コンパイラを使用して作成された ES1130 ターゲット用の旧バージョンのプロジェクトを ASCET-RP V5.5 で操作する場合、プロジェクトを開く時にはエラーメッセージは出力されませんが、コード生成時には、ASCET モニタウィンドウにエラーメッセージが出力されます。



また、ES1112 をターゲットとした旧バージョンのプロジェクトの場合は、プロジェクトを開く際に以下のようなエラーメッセージが表示され、ターゲットは PC に置き換えられます。



いずれの場合も、プロジェクトを開いた後に、ターゲットとコンパイラの適切な組合せを設定する必要があります。以下のように行ってください。

GNU コンパイラに変更する：

- プロジェクトエディタで、**Project Properties** ボタンをクリックします。

  “Project Properties” ウィンドウが開きます。
- “Build” ノードで、ターゲットを `ES1130` または `ES1135`、コンパイラを `GNU-C V3.4.4 (PowerPC)` に設定します。
- **OK** をクリックします。

  次に、プロジェクトで使用するターゲット用に C コードをコピーします。この手順は、26 ページの「オペレーティングシステムの設定と C コードをコピーする：」を参照してください。
- **Component → Touch → Recursive** を選択して、次回のコンパイル時にプロジェクトの全コンポーネントがコンパイルされるようにします。

オペレーティングシステムの設定と C コードをコピーする：

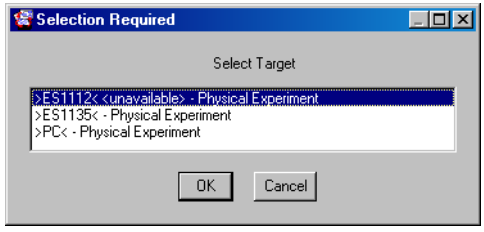
- プロジェクトエディタの“OS”タブを選択します。
- プロジェクトエディタで、**Operating System → Copy From Target** を選択します。

  “Selection Required”ダイアログボックスが開きます。

- “Selection Required”ダイアログボックスで PPC ターゲットを選択します。
- **OK** をクリックします。

  オペレーティングシステムのコードが、元のターゲットから ES1130 または ES1135 用にコピーされて変換されます。
- プロジェクトエディタで、**Extras → Copy C-Code From** を選択します。

  “Selection Required”ダイアログボックスが開きます。



- “Selection Required”ダイアログボックスで、元のターゲットと実験を選択します。
- **OK** をクリックします。

  元の C コードが、新しいターゲットと実験の組合せ用にコピーされ、変換されます。

2.4.3 Diab Data コンパイラへの変更

TIP Exp V4.2 までは、MS-DOS 版の Diab Data 4.1a コンパイラが 製品パッケージに含まれていました。このコンパイラは、ASCET-RP V5.5 においても ES1130 ターゲット用に使用することができます（`Diab Data V4.1a` を選択してください）が、ASCET-RP の製品パッケージには含まれていないため、別個に購入してインストールする必要があります。

注記

ES1135 ターゲットには Diab Data コンパイラは使用できません。

3 ASCET-RP を使用する際のヒント

3.1 旧バージョンのデータベースの使用

V4.1.1 より前のバージョンの ASCET で作成された ASCET データベースを ASCET V5.5 用データベースに変換するには、前もってそのデータベースを ASCET V4.1.1 で読み込み、保存しておく必要があります。

注記

非常に古いバージョンの ASCET は、プロジェクト（TIPExp V3.x またはそれ以前のプロジェクトや、PPC ターゲットのプロジェクトなど）の変換についての詳しい情報は、『TipExp V4.4 ユーザーガイド』を参照してください。

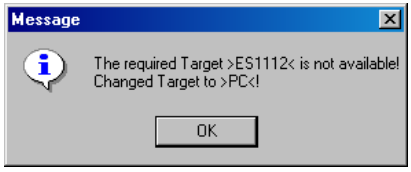
3.2 ES1000.1 用プロジェクトから ES1000.2/ES1000.3 用プロジェクトへの変換

実験システム ES1000.1 用に作成された ASCET プロジェクトを ES1000.2/ ES1000.3 で使用するには、プロジェクトの変換が必要です。以下の手順で変換を行ってください。

ES1000.1 用 ASCET プロジェクトを ES1000.2/ES1000.3 用に変換する：

- 変換する ASCET プロジェクトをロードします。

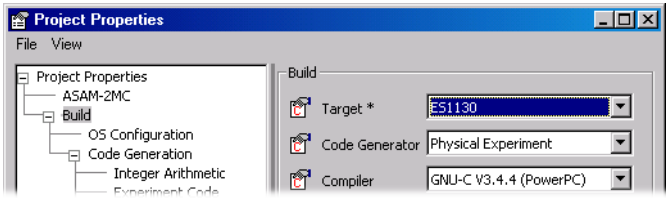
  ES1112 ターゲットは使用できない、という内容のメッセージが表示されます。



- メッセージを確認して、**OK** をクリックします。

  プロジェクトが開きます。使用できないターゲットの変わりに、PC ターゲット（>PC<）が選択された状態になっています。

- プロジェクトエディタの **Specify Code Generation Options** ボタンをクリックします。



- “Project Properties” ウィンドウの “Build” ノードで、以下のように各オプションを設定します。

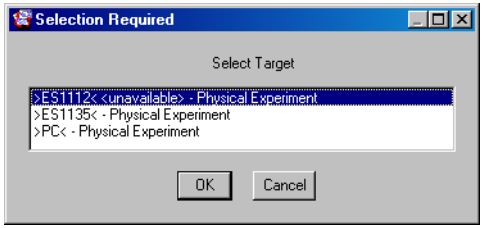
  `Target: >ES1130<` または `>ES1135<`
  `Compiler: GNU-C (PowerPC)`

- **OK** をクリックして “Project Properties” ウィンドウを閉じます。
- プロジェクトエディタの “OS” タブを開きます。
- プロジェクトエディタのメニューから **Operating System → Copy From Target** を選択します。

- “Selection Required” ダイアログボックスで元のターゲットを選択し、**OK** をクリックします。

  オペレーティングシステムのコードが、ES113x ターゲット用にコピーされて変換されます。

- プロジェクトエディタのメニューから **Extras → Copy C-Code From** を選択します。



- “Selection Required”ダイアログボックスで、元のターゲットと実験の組合せを選択します。
- **OK** をクリックします。

  元の C コードが、新しいターゲットと実験の組合せ用にコピーされ、変換されます。

これで、プロジェクトは ES1000.2/ES1000.3 システム用として使用できるようになりました。

3.3 dT の使用方法

ES113x ターゲットで実行されるプロトタイピングソフトウェアには、ERCOS^{EK} オペレーティングシステムが統合されます。ERCOS^{EK} の環境下においては、タスクの前回の実行開始時から今回の実行開始時までに経過した時間を表す「dT」という変数を利用できます。dT はタスクごとに存在し、それを使用するタスク自身についての値が常に格納されています（図 3-1）。

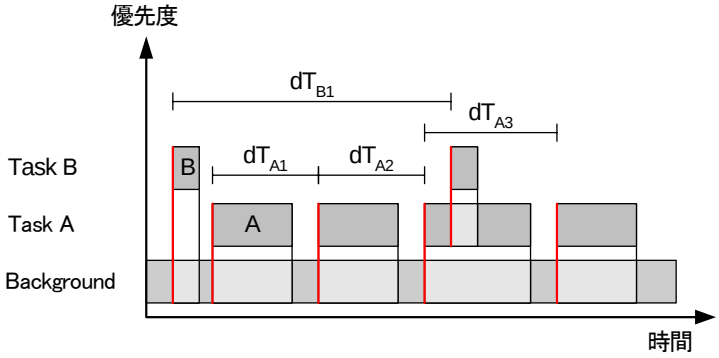


図 3-1 各タスク用の dT

ERCOS^{EK} において、dT は uint32 型のグローバル変数です。ERCOS^{EK} のヘッダファイルのうちのひとつで宣言され、現在実行中のタスクについての値をシステムクロックの単位で格納します。

dT を利用するには、ASCET のエディタ上の **dT** ボタンを使用して、秒単位の時間が格納される ASCET エレメント（real64）を作成します。

この dT は、ユーザーによって C コードエディタ上で宣言されていない状態で使用された場合でも、ERCOS^{EK} ファイル内で dT が宣言されているため、エラーメッセージは出力されません。しかし ERCOS^{EK} 内の dT と ASCET 内の dT では単位が異なるため（ERCOS^{EK} ではシステムクロック、ASCET では秒）、正しい値は得られません。必ず **dT** ボタンでエレメントを作成してから使用するようにしてください。

4 ラピッドプロトタイピング実験

この章では、ラピッドプロトタイピングの実験を行うためのいくつかの方法を紹介します。

4.1 ASCET を使用した実験

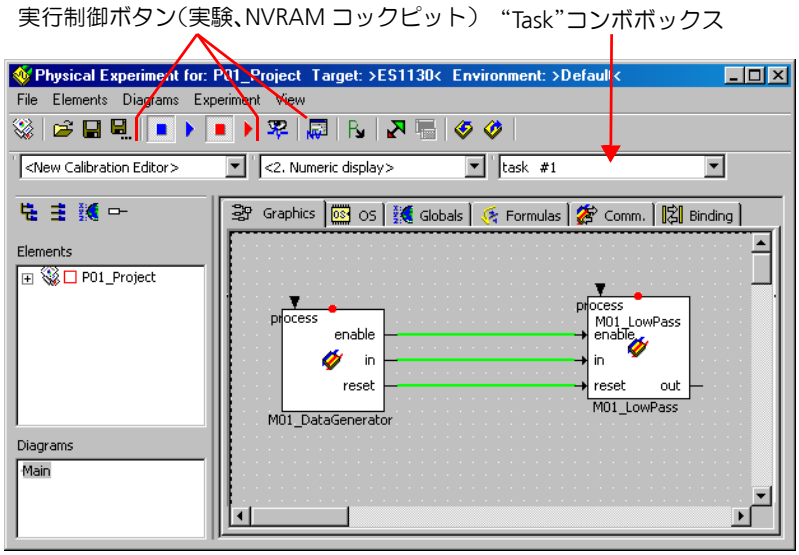
ASCET でラピッドプロトタイピングの実験を行う際は、プロジェクトエディタでオンライン実験とオフライン実験のいずれかを選択します。詳しい操作方法は、『ASCET ユーザーズガイド』の「プロジェクトの実験」の項を参照してください。また ASCET の実験環境については『ASCET ユーザーズガイド』の「実験環境」の項を参照してください。

ここではオンライン実験に固有な機能についてのみ説明されています。

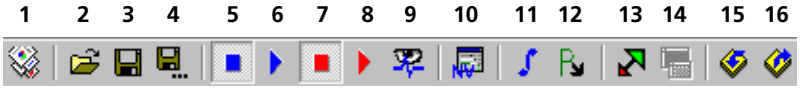
4.1.1 ユーザーインターフェース

オンライン実験のユーザーインターフェースは、オフライン実験の場合とほとんど同じです。異なるのは以下の点のみです。

- プログラムの実行を制御するボタン
- NVRAM コックピット（ES1135 を使用する場合のみ）
- “Task” コンボボックス
- **Experiment** メニュー



ツールバーのボタン



1. Exit to Component - 実験を終了してプロジェクトエディタに戻る
2. Load Environment - 実験環境（測定／適合ウィンドウの設定と、各ウィンドウに割り当てられた変数）のロード
3. Save Environment - 現在の実験環境の保存
4. Save Environment As - 現在の実験環境を任意の名前で
5. Stop ERCOS - オペレーティングシステムの実行終了（= 実験の終了）
6. Start ERCOS - オペレーティングシステムの実行開始（= 実験の開始）
7. Stop Measurement - 測定の終了（= データ表示の終了）
8. Start Measurement - 測定の開始（= データ表示の開始）
9. Open Data Logger - データロガーを開く
10. Open NVRAM Cockpit - NVRAM コックピットを開く（73 ページ参照）
    このボタンは ES1135 がターゲットとして選択されている場合のみ表示されます。
11. Open CT Solver - 積分メソッドの設定ダイアログボックスを開く
    このボタンは CT ブロックプロジェクトまたはハイブリッドプロジェクトを実行する際にのみ表示されます。
12. Update Dependent Parameters - 依存パラメータの値の更新
13. Expand / Collapse Window - コンポーネントの表示／非表示切り替え
14. Always on top - 実験ウィンドウを常に最前面に表示
15. Navigate down to child component - 選択された、内包されるコンポーネントの内容を表示
16. Navigate up to parent component - 上位のコンポーネントを表示

Experiment *メニュー*

以下のメニューコマンドが含まれます。

- Data Logger
  データロガーを開きます。
- NVRAM Cockpit
  （ES1135 がターゲットとして選択されている場合のみ使用できます）
  NVRAM コックピットを開きます。
- Stop ERCOS
  オペレーティングシステムの実行を終了します。
- Start ERCOS
  オペレーティングシステムの実行を開始します。

— Stop Measurement

測定を終了します。

— Start Measurement

測定を開始します。

— Open Target Debugger

C コードコンポーネント用のデバッガウィンドウを開きます。

— Update Calibration Windows

適合ウィンドウの内容を更新します。

— Close Calibraion Windows

すべての適合ウィンドウを閉じます。

— Close Measure Windows

すべての測定ウィンドウを閉じます。

ユーザーインターフェースの他のエレメントは、オフライン実験の場合と同じです。これらは『ASCET ユーザーズガイド』の「実験環境」の項に説明されています。

4.1.2 オンライン実験の実行

プロジェクトのオンライン実験は、プロジェクトエディタから起動します。

オンライン実験を開始する：

- プロジェクトまたはコンポーネントを開きます。
- コンポーネントの実験を行う場合は、デフォルトプロジェクトを開きます。
- プロジェクトプロパティウィンドウを開いて“Build”ノードを選択し、`ES1130` または `ES1135` をターゲットとして選択します。

  “Experiment Target”コンボボックスで、`Offline (RP)` または `Online (RP)` を選択できるようになります。

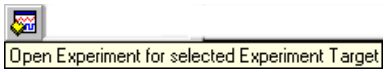
  また、**Open Experiment for selected Experiment target** ボタンと **Reconnect to Experiment of selected Experiment target** ボタンが有効になります。

- “Experiment Target”コンボボックスから `Online (RP)` を選択します。

  `Offline (RP)` は、ターゲット上でオフライン実験を行う際に選択します。

- **Component → Open Experiment** を選択します。

または



- **Open Experiment for selected Experiment target** ボタンをクリックします。

ハードウェアオプション（2.1 項参照）の **Use ETAS Network Manager (enables 'Select Hardware')** オプションをオンに設定しておくと、状況に応じてハードウェア選択ダイアログボックス（2.2.1 項参照）が自動的に開きます。

ハードウェアを選択する（ETAS ネットワークマネージャを使用）：

注記

ETAS ネットワークマネージャを使用しない場合、この操作は不要です。36 ページ「エラーが発生した場合：」以降をお読みください。

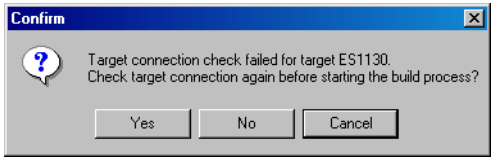
- "Select simulation board of type <*type*>" フィールドで、使用したいハードウェアを選択します。

  OK ボタンが使用可能になります。
- 必要に応じて他の設定を行います。
- **OK** をクリックしてダイアログボックスを閉じます。

  システムは、選択されたハードウェアが使用可能であるか、またそのハードウェアが、プロジェクトのコード生成オプションで選択されているターゲットと一致するかをチェックします。

エラーが発生した場合：

選択されたハードウェアが実際に存在しない場合、以下のエラーメッセージが表示されます。



- **Yes** をクリックして、ハードウェア検索を再実行します。

または

- **No** をクリックして、ハードウェアが未接続のままビルド処理を実行します。

  ETAS ネットワークマネージャを使用している場合、ビルド処理の終了後に、再度ハードウェア選択ダイアログボックスが開きます。

  ETAS ネットワークマネージャを使用していない場合は、ビルド処理の終了後、再度ハードウェア検索を行うかどうかが尋ねられます。

または

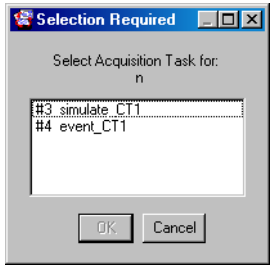
- **Cancel** をクリックして、実験開始を中止します。

ETAS ネットワークマネージャを使用していない場合、またはハードウェアの選択（36 ページ参照）が正常に行われた場合、実験が開始されると（35 ページ参照）、すぐにコンポーネントのデフォルトの実験環境が開きます。

オンライン実験用に実験環境を開く：

- 複数の環境がすでに保存されている場合は、どれを開くかを選択します。詳細は、『ASCET ユーザーズガイド』の「環境設定のロードと保存」という項を参照してください。

  プロジェクト内に複数のタスクが含まれている場合、各測定変数について、その値を取得するためのタスクを選択する必要があります。



- “Selection Required”ダイアログボックスでタスクを選択し、**OK** をクリックします。

  測定タスクは、後から“Task”コンボボックスで選択することができます。

オンライン実験に必要な準備は、測定ウィンドウと適合ウィンドウの用意だけです。これらのウィンドウの使用方法は、オフライン実験の場合と同じです。『ASCET ユーザーズガイド』の「実験環境」の項を参照してください。

これらの準備ができたら、実験を開始します。オンライン実験の実行中には、測定／適合ウィンドウの表示オプションの変更や、これらのウィンドウの追加や削除を行え、また適合システムを利用してパラメータ（適合変数）の値を変更することができます。

オンライン実験と測定を開始する：

- 実験を行うプロジェクトの実験環境を開きます。
- **Experment → Start ERCOS** を選択します。

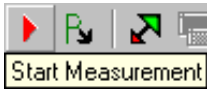
または

- **Start ERCOS** ボタンをクリックします。

  オペレーティングシステムと実験が起動されます。ここでは、まだ測定データは表示されません。

- **Experment → Start Measuremet** を選択します。

または

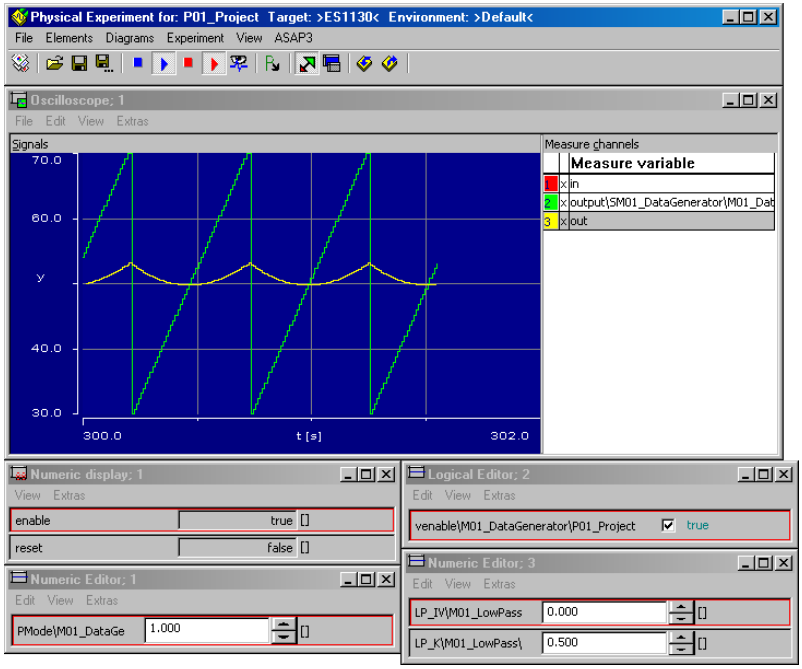


- **Start Measurement** ボタンをクリックします。

注記

測定を実行すると、モデルのリアルタイム挙動が影響を受ける可能性があります。

測定が開始され、測定システムで設定されたすべての変数の値が所定のウィンドウに表示されます。



測定を終了する：

- **Experment → Stop Measurement** を選択します。

または

- **Stop Measurement** ボタンをクリックします。

  測定が終了しますが、実験はそのまま続行されます。測定が再開されると、時間軸は現在の値に更新されます。

  すべての設定は保持されます。測定データはオシロスコープウィンドウにそのまま表示されているので、そのデータを分析することができます。

実験を終了する：

- **Experment → Stop ERCOS** を選択します。

または

- **Stop ERCOS** ボタンをクリックします。

  測定が行われていた場合は、測定も終了します。オペレーティングシステムの実行が終了するため、実験も終了します。この状態、つまり「起動していない状態」のオペレーティングシステムには、Init タスクを 1 つ割り当てることができます。このタスクはオペレーティングシステムが停止している間に実行されます。このタスクを利用して、外部ハードウェアのリセットなどを行うことが可能です。オペレーティングシステムが再起動すると、開始モードが実行され、その `Init` タスクが再度実行されます。オペレーティングシステムをポーズする機能はありません。

- **Exit to Component** ボタンをクリックして実験環境を閉じ、プロジェクトエディタに戻ります。

または

- **File → Exit** を選択して実験環境を閉じ、プロジェクトを閉じます。

オフライン実験の場合と同様に、実験が実行されているがどうかに関わらず、プロジェクトのインプリメンテーションを実験環境からることができます。

またオフライン実験と同様、C コードで記述されたコンポーネントについては、実験中にデバッグ情報やエラーメッセージを表示することができます。これらのメッセージは C コード内に任意に埋め込むことができます。デバッグ情報は、実験中に開くことのできる「ターゲットデバッグビューア」に表示され、エラーメッセージは ASCET モニタウィンドウに出力されます。詳しくは、『ASCET ユーザーズガイド』の「実験の実行」の項を参照してください。

4.1.3 スタンドアロンモード

実験ターゲットがあれば、実験環境を使用せずにスタンドアロンモードでコードを実行することができます。ただしその場合、実験ターゲットにフラッシュメモリが搭載されていて、そこから実験ターゲットを起動できる、という条件が満たされている必要があります。

実験を FLASH メモリにロードする：

- “Experiment Target”コンボボックスから`Online (RP)` を選択します。
- **Component → Flash Target** を選択します。

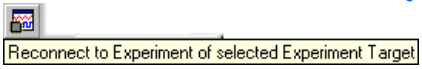
  ハードウェアオプションの **Use ETAS Network Manager (enables 'Select Hardware')** オプション（2.1 項を参照）がオンになっていると、状況に応じてハードウェア選択ダイアログボックス（2.2.1 項を参照）が開きます。
  このダイアログボックスの使用方法は、36 ページ「ハードウェアを選択する（ETAS ネットワークマネージャを使用）：」に説明されています。

  ASCET で生成されたコードが実験ターゲットの FLASH メモリに書き込まれます。このコードには、FRASH メモリから起動するためのスタートアップルーチンが組み込まれます。これ以降、ターゲットハードウェアは、リセットが行われるたびに ASCET モデルを実行します。

スタンドアロンモードで実験を行う：

- **Component → Reconnect To Experiment** を選択します。

または



- **Reconnect to Experiment of selected Experiment Target** ボタンをクリックします。

  ハードウェアオプションの **Use ETAS Network Manager (enables 'Select Hardware')** オプション（2.1 項を参照）がオンになっていると、状況に応じてハードウェア選択ダイアログボックス（2.2.1 項を参照）が開きます。
  このダイアログボックスの使用方法は、36 ページ「ハードウェアを選択する（ETAS ネットワークマネージャを使用）：」に説明されています。

  オンライン実験環境が起動し、実験が初期状態から開始されます。

実験ターゲット上で通常のモード（非スタンドアロンモード）で実行されている実験プログラムへの接続を断って、その後、その実験に再接続することもできます。実行中の実験プログラムとの接続を断つには実験環境を終了するだけでよく、実験ターゲット上のプログラムを終了する必要はありません。

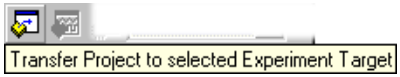
4.2 INCA を使用した実験

ASCET プロジェクトの実験は、INCA（バージョン 4.0.4 以降、またそのバージョンに対応するアドオン製品 INCA-EIP が必要）を使用して行うことができます。このためには、プロジェクトエディタで、プロジェクトを INCA に転送します。

プロジェクトの転送を開始する：

- 実験を行うプロジェクトを開きます。
- プロジェクトプロパティウィンドウの“Build”ノードで、ターゲットとして `ES1130` または `ES1135` を選択し、コンパイラに `GNU-C V3.4.4 (PowerPC)` を選択します。
- “Experiment Target”コンボボックスから `INCA` を選択します。

  Transfer Project to selected Experiment Target および **Reconnect to Experiment of selected Experiment Target** ボタンが有効になります。

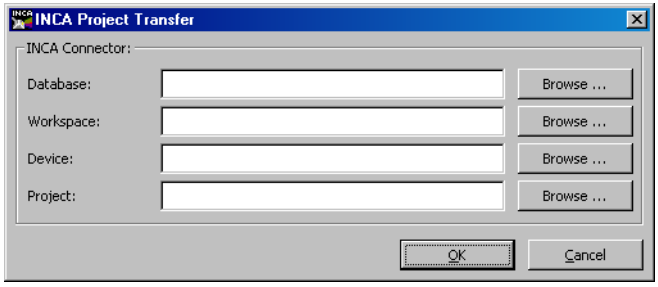


- **Transfer Project to selected Experiment Target** ボタンをクリックします。

または

- **Component → Transfer Project** を選択します。

  “INCA Project Transfer”ダイアログボックスが開きます。



このダイアログボックスで、INCA データベースと、その中に含まれるワークスペースとプロジェクトを指定します。

この時点で INCA が起動していない場合、いずれかの **Browse...** ボタンまたは **OK** ボタンをクリックすると INCA が起動します。もしも複数のバージョンの INCA が同じ PC にインストールされている場合は、最後にインストールされたバージョン（最新のバージョンとは限りません）が起動します。

注記

ここで、V3.x またはそれ以前のバージョンがインストールされていたり、または INCA がまったくインストールされていない場合、エラーメッセージが表示されてデータ転送は中止されます。

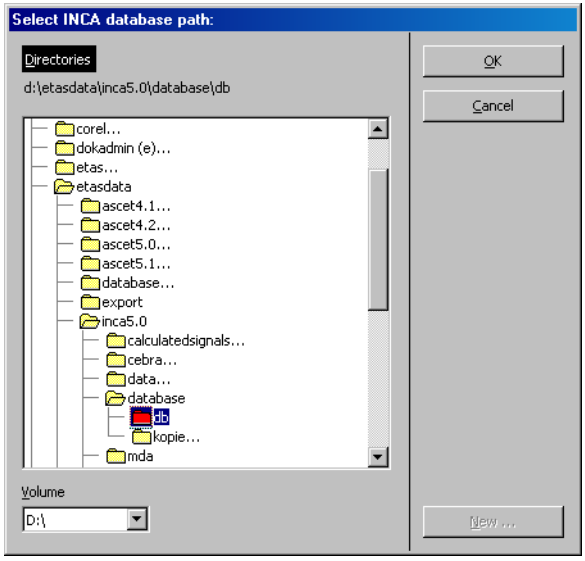
INCA データベースのパスを指定する：

- “INCA Project Transfer”ダイアログボックスの“INCA Database”フィールドに既存のデータベースパスを入力します。

または

- **Browse** ボタンをクリックしてデータベースパスを検索します。

  “Select INCA database path”ダイアログボックスが開きます。データベースは赤いフォルダシンボルで示されます。



- INCA データベースを選択して **OK** をクリックします。

注記

このダイアログボックスでは、ASCET データベースと INCA データベースがまったく同様に表示されますが、ここでは必ず **INCA データベース**を選択してください。

または

- **New** ボタンをクリックして、プロジェクトの転送先となる新しい INCA データベースを作成するための空のディレクトリを作成します。作成されたディレクトリは黄色のフォルダとして表示されます。

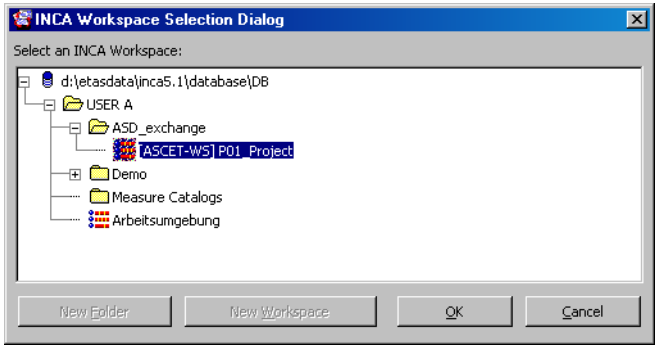
INCA ワークスペースを選択する：

- “INCA Project Transfer”ダイアログボックスの“INCA Workspace”フィールドに、既存のワークスペースのパスと名前を入力します。

または

- **Browse** ボタンをクリックしてワークスペースを検索します。

  “INCA Workspace Selection Dialog”ダイアログボックスが開き、選択されている INCA データベースに含まれるワークスペースの一覧が表示されます。



- 既存のワークスペースを選択します。

または

- 既存のフォルダまたは新しいフォルダ内に、新しいワークスペースを作成します。手順は次に示されています。

- ワークスペースを指定したら、**OK** をクリックします。

“Workspace Selection Dialog”ダイアログボックスで新しいフォルダ／ワークスペースを作成する：

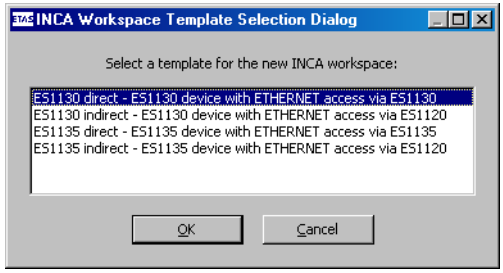
1. フォルダを作成する
   - “INCA Workspace Selection Dialog”ダイアログボックスでデータベースまたはフォルダを選択します。

     New Folder ボタンが有効になります。
   - **New Folder** ボタンをクリックします。
   - “Create New INCA Folder”ダイアログボックスにフォルダ名を入力して **OK** ボタンをクリックします。

     新しいフォルダが作成されます。
2. ワークスペースを作成する
   - フォルダを選択します。

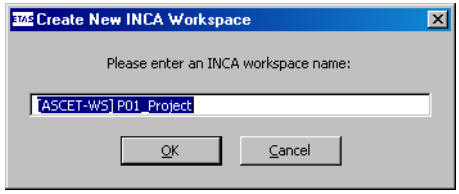
     New Workspace ボタンが有効になります。
   - **New Workspace** ボタンをクリックします。

     “INCA Workspace Template Selection Dialog”ダイアログボックスが開きます。



- ワークスペースのテンプレートを選択して **OK** をクリックします。

  “Create New INCA Workspace” ダイアログボックスが開きます。



  `[ASCET-WS] <ascet project name>` というデフォルト名が割り当てられます。

- ワークスペースの名前を編集して、**OK** をクリックします。

  選択されたフォルダに新しいワークスペースが作成され、そのワークスペースが選択された状態となります。

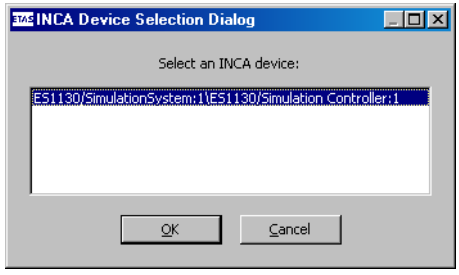
INCA デバイスを選択する：

- “INCA Project Transfer” ダイアログボックスの “INCA Device” フィールドにデバイスを入力します。

または

- **Browse** ボタンをクリックしてデバイスを検索します。

  “INCA Device Selection Dialog” ダイアログボックスが開き、選択可能なデバイスの一覧が表示されます。



- デバイスを選択して **OK** をクリックします。

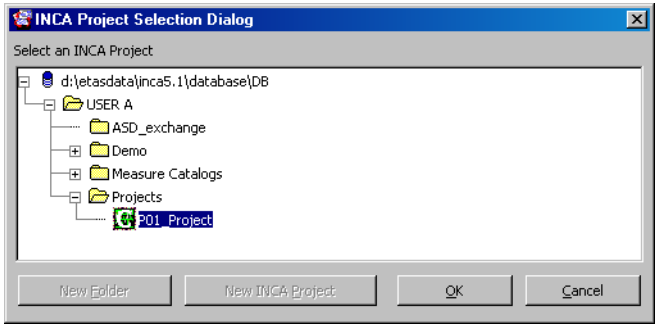
INCA プロジェクトを選択する：

- “INCA Project Transfer”ダイアログボックスの“INCA Project”フィールドに既存の INCA プロジェクトのパスを名前を入力します。

または

- **Browse** ボタンをクリックしてプロジェクトを検索します。

  “INCA Project Selection Dialog”ダイアログボックスが開き、選択されているデータベース内のプロジェクトが一覧表示されます。



- 既存のプロジェクトを選択します。

または

- 既存のフォルダまたは新しいフォルダ内に、新しいプロジェクトを作成します。手順は次に示されています。
- プロジェクトを指定したら、**OK** をクリックします。

注記

選択されたプロジェクトは、この転送処理によって作成されたプロジェクトに強制的に置き換わりますが、その際、警告メッセージは出力されません。

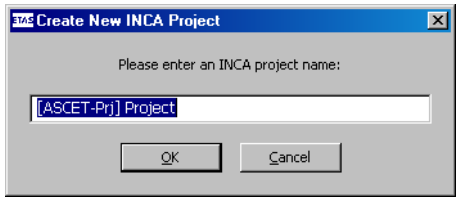
“INCA Project Selection Dialog”ダイアログボックスで新しいフォルダ／プロジェクトを作成する：

1. フォルダを作成する
   - 45 ページの「フォルダを作成する」の方法で、新しいフォルダを作成します。
2. プロジェクトを作成する

- フォルダを選択します。

  New INCA Project ボタンが有効になります。
- **New INCA Project** ボタンをクリックします。

  “Create New Workspace”ダイアログボックスが開き、`[ASCET-Prj] <ascet project name>` というデフォルト名が表示されます。



- プロジェクト名を入力して **OK** をクリックします。

  選択されたフォルダに新しいプロジェクトが作成され、そのプロジェクトが選択された状態となります。

データを転送する：

- “INCA Project Transfer”ダイアログボックスの項目を設定したら、**Continue** ボタンをクリックします。

  ASCET プロジェクトから INCA プロジェクトへの転送が実行されます。この際、最初に ASAM-MCD-2MC コードが生成され、必要に応じてプロジェクトの内容の更新（コード生成、コンパイル、リンク）が行われます。

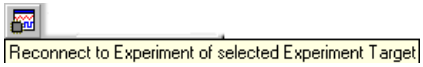
  “INCA Project”フィールドに既存の INCA プロジェクトが指定されていた場合、そのプロジェクトは上書きされます。

転送が終了すると、INCA での実験を開始することができます。INCA での実験についての詳しい情報は、『INCA ユーザーズガイド』および『INCA-EIP ユーザーズガイド』を参照してください。

ASCET でバックアニメーションを行う際は、特殊な実験環境を使用します。そこでは通常の方法でパラメータの適合を行うことができます。測定システムも通常どおりに動作しますが、ASCET のオフライン／オンライン実験とは異なり、オシロスコープ、レコーダ、およびデータロガーの機能は使用できません。INCA でのバックアニメーションでは、これらの機能に必要な同期測定が行えないためです。その他の測定インストゥルメントを使用して測定作業を行ってください。

バックアニメーションを行う：

- INCA で、プロジェクトの実験を開始します。
- ASCET プロジェクトエディタで、“Experiment Target”で INCA が選択されていることを確認してください。



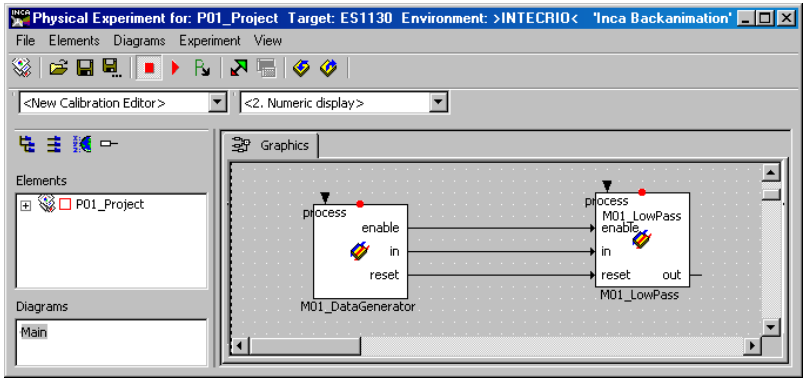
Reconnect to Experiment of Selected Experiment Target ボタンをクリックします。

または

- **Component → Reconnect to Experiment** を選択します。

ハードウェアオプションの **Use ETAS Network Manager (enables 'Select Hardware')** オプション（2.1 項を参照）がオンになっていると、状況に応じてハードウェア選択ダイアログボックス（2.2.1 項を参照）が開きます。
このダイアログボックスの使用方法は、36 ページ「ハードウェアを選択する（ETAS ネットワークマネージャを使用）：」に説明されています。

INCA で実験を実行するための接続が確立され、“Physical Experiment...”ウィンドウが開きます。タイトルバーに“INCA Backanimation”と表示され、バックアニメーション専用の実験環境であることがわかります。

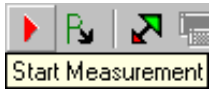


通常のオフライン／オンライン実験とは異なり、このウィンドウには“Graphics”タブのみが表示されます。

- 必要な測定ウィンドウを作成して設定します（『ASCET ユーザーズガイド』の「測定システム」の項を参照してください）。

- 必要な適合ウィンドウを作成して設定します（『ASCET ユーザーズガイド』の「適合システム」の項を参照してください）。
- **Experiment → Start Measuremet** を選択します。

または



- **Start Measurement** ボタンをクリックして測定を開始します。

  測定／適合ウィンドウの表示内容が周期的に更新されます。

実験のロード、保存、エクスポートの方法は、『ASCET ユーザーズガイド』の「環境設定のロードと保存」の項に説明されています。ロードされた環境内に、バックアニメーションで使用できないエレメント（オシロスコープなど）が含まれていると、それらのエレメントは無視されます。

ここでは、モニタ機能（『ASCET ユーザーズガイド』の「モニタ機能」の項を参照してください）を使用して、数値変数と論理変数をモニタすることもできます。モニタ機能の設定は、環境内に保存されます。

また、プロジェクト内のコンポーネント間をナビゲートすることができます（『ASCET ユーザーズガイド』の「ブロックダイアグラム間のナビゲート」の項を参照してください）。

プロジェクトにステートマシンが含まれている場合、ステートマシンのアニメーション機能（『ASCET ユーザーズガイド』の「ステートマシンの検証」の項を参照してください）を利用できます。

実験で得られたデータは、ASCET モデルやハードディスクに書き込むことができます。またデータをハードディスクから読み込むことも可能です。詳しくは『ASCET ユーザーズガイド』の「データの扱い」の項を参照してください。

バックアニメーションを終了する：

- File → Exit を選択します。

または

- **Exit to Component** ボタンをクリックします。

  バックアニメーションが終了し、実験環境が閉じます。INCA の実験はそのまま続行されます。

4.3 INTECRIO を使用した実験

ASCET-RP と INTECRIO がインストールされていると、ラピッドプロトタイプ実験を INTECRIO を使用して行うことができます。このためには、プロジェクトエディタのメニューコマンドを利用して、プロジェクトを INTECRIO に転送します。

この項には、INTECRIO を使用して実験を行う際の一般的な方法がまとめられています。具体的な操作例は、11.1「チュートリアル - INTECRIO を使用した実験」の項に紹介されています。

INTECRIO で実験を行うには、まず始めに ASCET プロジェクトを作成します。この際、以下の点に注意が必要です。

- デフォルト状態において、ASCET 内で受信のみが行われるメッセージ、つまり対応する送信メッセージが存在しない受信メッセージは、INTECRIO 内で「シグナルシンク」となります。またその逆に ASCET 内で送信のみが行われるメッセージ、つまり対応する受信メッセージが存在しない送信メッセージは、INTECRIO 内で「シグナルソース」となります。ASCET 内で送受信の両方が行われるメッセージについては、INTECRIO への統合時には省略されます。
- プロジェクトに未解決のメッセージ（対応するエクスポートメッセージがないインポートメッセージなど）がある場合、INTECRIO 用のコード生成時にエラーメッセージが出力されます。

  この場合、自動的にメッセージを解決するか、またはコード生成をキャンセルして手動操作でメッセージを解決します。
- グローバル変数とパラメータを使用することはできますが、これらの使用は極力避けるようにしてください。
- プロジェクトのビルドオプションで、`Prototyping` というターゲットを選択します。このターゲットを選択すると、プロジェクトエディタの“Experiment Target”コンボボックスに `INTECRIO` というアイテムが表示されます。このターゲット用に生成されるコードは、INTECRIO でサポートされているすべての実験ターゲットで使用できます。

  この際、ターゲットに `ES1130` または `ES1135` を選択することもできますが、INTERIO に実験を転送するには、`Prototyping` を選択するようにしてください。
- `Prototyping` ターゲットが選択されていると、プロジェクトエディタの“OS”タブにおいて以下の設定項目が無効になります。
  - “Preemp. Levels”フィールドと“Coop. Levels”フィールド（全タスク用）
  - **Enable Monitoring** オプション、“pre-/post hooks”コンボボックス（全タスク用）
  - “ISR Source”フィールドと“Min. Period”フィールド（Interrupt タスクのみ）
  - “Max. Number of Activation”フィールド（Alarm タスクと Software タスクのみ）
  - **Autostart** オプション（Alarm タスクと Software タスクのみ）

  Operating System → Copy From Target コマンドで OS 設定を `Prototyping` ターゲットにコピーすると、上記の設定はコピーされません。

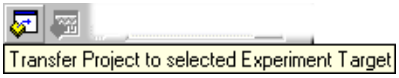
プロジェクトが完成したら、コード生成を行うために、まずそのプロジェクトを INTECRIO に転送します。

転送の設定を開始する：

- 実験を行うプロジェクトを開きます。
- プロジェクトプロパティウィンドウの“Build”ノードで、`Prototyping` ターゲットが選択されていることを確認してください。

  “Experiment Target”コンボボックスで、`INTECRIO` が選択された状態となります。

  Transfer Project to selected Experiment Target および **Recconect to Experiment of selected Experiment Target** ボタンが有効になります。

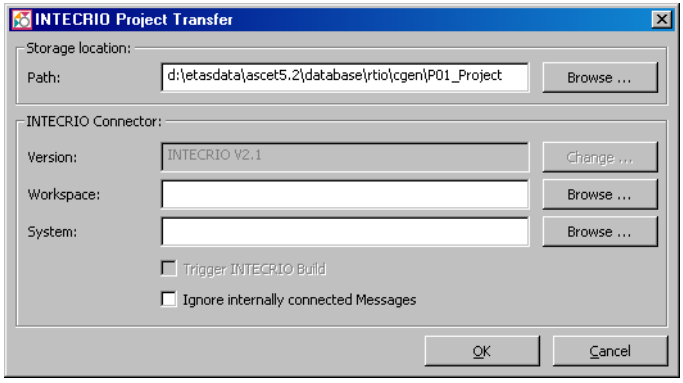


- **Transfer Project to selected Experiment Target** ボタンをクリックします。

または

- **Component → Transfer Project** を選択します。

  “INTECRIO Project Transfer”ダイアログボックスが開きます。“Path”フィールドには、デフォルトの転送先パス（プロジェクト名を含みます）が表示されます。このディレクトリはデータベースパス内に位置しているため、**Keep files in Code generation Directory** オプションがオフになっていても、ファイルが自動的に削除されることはありません。



この“INTECRIO Project Transfer”ダイアログボックスボックスで、生成されたファイルを格納するパスを指定します。それには以下の 4 通りの方法があります。

- INTECRIO に必要なコード生成のみを行うには、“Path”以外のフィールドは空白のままにしておきます。

  すでに生成されたコードを INTCRIO に転送する場合は、この方法で行います。

  注記

  INTECRIO が PC にインストールされていなくても、INTECRIO 用のコード生成を行うことは可能です。

- コード生成を行い、それを INTECRIO にインポートするには、さらに INTECRIO のバージョンを選択して INTECRIO のワークスペースを指定する必要があります。“Systems”フィールドは空白のままにします。

  デフォルトでは、“Version”フィールドには最後にインストールされた INTECRIO のバージョンが表示されます。1 つのバージョンしかインストールされていない場合はそのバージョンが表示され、**Change** ボタンは無効になります。

  指定したワークスペースが存在しない場合は、自動的に作成されます。

- コード生成を行って INTECRIO にインポートし、それを INTECRIO に統合する（つまり INTECRIO のシステムプロジェクトに追加する）には、INTECRIO システムプロジェクトを指定します。

  この場合、指定したワークスペースとシステムプロジェクトはすでに存在していなければなりません。

- コード生成を行って INTECRIO にインポートし、それを INTECRIO に統合して、さらに INTECRIO でビルド処理を開始するには、すべてのフィールドを設定して **Trigger INTECRIO Build** をオンにします。

  ビルド処理が正常に実行され、有効なプロトタイプが生成されるようにするには、INTECRIO においてハードウェアシステムとオペレーティングシステムコンフィギュレーションが完全に設定されている必要があります。

注記

ASCET は、指定されたワークスペースがどのバージョンの INTECRIO で作成されたかをチェックしません。
ワークスペースを作成した際に使用したものと異なるバージョンを選択すると、転送処理は正常に行われません。

ASCET モデル内で送受信されるメッセージを INTECRIO のシグナルソースとして利用できるように変換する必要がある場合は、**Ignore internally connected messages** オプションをオフにしてください。このオプション設定は、上記の 4

通りの転送方法のすべてにおいて適用されます。表 4-1 は、各タイプのメッセージについて、オプションをオン／オフに設定した場合にどのような INTECRIO インターフェースに変換されるかを示したものです。

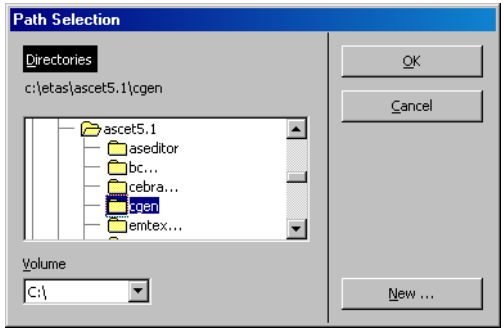
メッセージアクセスをアクセスする ASCET コンポーネント			INTECRIO インターフェース	
プロジェクト	モジュール A	モジュール B	オプション = オン	オプション = オフ
S/R			—	—
S/R	S		シグナルソース	シグナルソース
S/R	R		シグナルシンク	シグナルシンク
S/R	S/R		シグナルソース	シグナルソース
	S		シグナルソース	シグナルソース
	S	S	シグナルソース	シグナルソース
	S	R	—	シグナルソース
	S	S/R	—	シグナルソース
	R		シグナルシンク	シグナルシンク
	R	R	シグナルシンク	シグナルシンク
	R	S/R	—	シグナルソース

表 4-1 メッセージ変換規則（S：コンポーネントによって送信されるメッセージ、R：コンポーネントによって受信されるメッセージ）

生成されたファイルを格納するパスを指定する：

- “INTECRIO Project Transfer“ダイアログボックスで、“Path”フィールドの隣の **Browse** ボタンをクリックします。

  “Path Selection”ダイアログボックスが開きます。



- 必要に応じて“Volume”コンボボックスでボリュームを選択します。
- “Directories”リストから既存のディレクトリを選択します。

または

- **New** ボタンをクリックして新しいディレクトリを作成します。
- **OK** をクリックします。

  “Path”フィールドにパスが表示されます。この設定は、プロジェクト内に保存され、次回の転送時には同じ設定がプリセットされます。

PC 上に INTECRIO の 1 つのバージョンしかインストールされていない場合は、そのバージョンが自動的に選択されるので、次のバージョン選択操作は無効になります。

INTECRIO のバージョンを選択する：

- “INTECRIO Project Transfer”ダイアログボックスで、“Version”フィールドの隣の **Change** ボタンをクリックします。

  選択ダイアログボックスが開き、PC 上にインストールされているすべての INTECRIO のバージョンが表示されます。
- バージョンを選択します。
- **OK** をクリックします。

  “Version”フィールドにバージョンが表示されます。この設定は、プロジェクト内に保存されます。

INTECRIO ワークスペースを選択する：

- “INTECRIO Project Transfer”ダイアログボックスの“Workspace”フィールドに、使用する INTECRIO ワークスペースのパスと名前を入力します。

または

- “Workspace”フィールドの隣の **Browse** ボタンをクリックします。

  Windows のファイル選択ダイアログボックスが開きます。
- 使用するワークスペースを含むディレクトリを選択します。

- ワークスペース（*.iow）を選択します。
- **Open** をクリックします。

  “Workspace”フィールドにワークスペース名が表示されます。

ASCET プロジェクトを INTECRIO システムプロジェクトに正しく統合するには、ワークスペースが選択されていて、かつ INTECRIO が起動している必要があります。

INTECRIO システムプロジェクトを選択する：

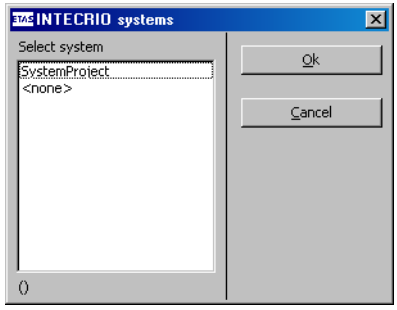
- “INTECRIO Project Transfer”ダイアログボックスの“System”フィールドに、使用する INTECRIO システムプロジェクトの名前を入力します。

または

- “System”フィールドの隣の **Browse** ボタンをクリックします。

  INTECRIO が起動していない場合は、ここで起動されます。

  “INTECRIO systems”ダイアログボックスが開きます。ワークスペースに含まれるすべてのシステムプロジェクトが表示されます。



- ASCET プロジェクトを追加したシステムプロジェクトを選択します。
- **OK** をクリックします。

  “System”フィールドにシステムプロジェクト名が表示されます。

INTECRIO システムに 1 つの ASCET プロジェクトのみが含まれ、かつハードウェアシステムと OS コンフィギュレーションが INTECRIO 内で設定されている場合、プロジェクトの転送後、INTECRIO 内でビルド処理が自動的に開始されます。

INTECRIO のビルド処理を選択する：

- **Trigger INTECRIO Build** オプションをオンにします。

  この操作は、INTECRIO ワークスペースとシステムプロジェクトが選択されている場合にのみ可能です。

最後に、INTECRIO への転送を実行します。

転送を実行する：

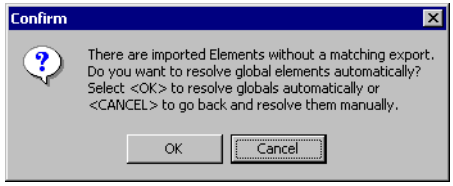
- “INTECRIO Project Transfer” ダイアログボックスで必要な項目をすべて設定したら、**OK** をクリックします。

  ASCET プロジェクトの INTECRIO への転送が開始されます。

  転送時の状況に応じて、以下のようなワーニングメッセージが表示される場合があります。

1. 未解決のメッセージが存在する場合：

   プロジェクトに未解決のメッセージが含まれていると、以下のダイアログボックスが開きます。



- **OK** をクリックすると、メッセージは自動的に解決されます。

  自動処理が完了すると、INTECRIO への転送が続行されます。

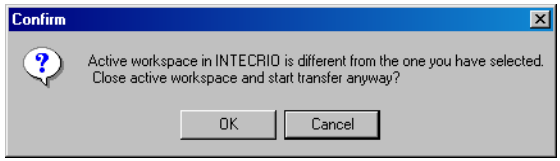
または

- コード生成を中断してマニュアル操作でメッセージの解決を行うには、**Cancel** をクリックします。

  この場合、転送を再度実行する必要があります。

2. 別の INTECRIO ワークスペースが開いている場合：

   INTECRIO が起動していて、指定されたものとは異なるワークスペースが開いていた場合、転送開始時に以下のダイアログボックスが開きます。



   - **OK** をクリックすると、開いているワークスペースが閉じて、転送処理が実行されます。

     自動処理が完了すると、INTECRIO への転送が続行されます。

   または

   - 転送を中止するには **Cancel** をクリックします。

プロジェクトの転送中には、INTECRIO での作業に必要なすべてのファイルが生成され、指定されたディレクトリに保存されます。すべてのオプションが設定されていれば、INTECRIO が起動し、プロジェクトが INTECRIO にインポートされてシステムプロジェクトに統合され、INTECRIO のビルド処理が開始されます。

INTECRIO 用に作成されるファイルには、以下のようなものがあります。

- `<project name>.six`

  XML ベースの SCOOP-IX という言語でプロジェクトインターフェースが記述された、ディスクリプションファイルです。
  ハードウェアコンフィギュレーションの設定は INTECRIO 内で行われるため、使用する HWC モジュールのインターフェースは SCOOP-IX ファイルには出力されません。

  SCOOP-IX のインターフェースディスクリプションには、以下のような情報が含まれます。

  — C 変数の名前、型、サイズ

  — C 関数の名前、戻り値、引数

  — 各 C エレメントが含まれるファイル

  詳しくは、『INTECRIO ユーザーズガイド』の「SCOOP と SCOOP-IX」の章を参照してください。

  OS エディタでオペレーティングシステムの設定が行われている場合、SCOOP-IX ファイル（V1.1）には、OS に関する情報のうち、無効になっている情報（51 ページ参照）以外のものが出力されます。ただし、この情報は現バージョンの INTECRIO では使用されません。

- `<project name>.a2l`

  INTECRIO 用に作成された ASAM-MCD-2MC ファイルです。

  このファイルにも HWC モジュールは含まれません。

- `<project name>.oil`

  INTECRIO で使用されるオペレーティングシステムのディスクリプションが格納されたファイルです。
  なおハードウェアと OS のコンフィギュレーション設定は INTECRIO 内で行われるため、HWC モジュールはこのファイルには含まれません。

  注記

  このファイルは、自動的には INTECRIO にインポートされません。INTECRIO でマニュアル操作を行ってオペレーティングシステムを設定するか、または `*.oil` ファイルをマニュアル操作でインポートしてください。
  この `*.oil` ファイルのフォーマットは、OSEK 規格には準拠していません。このファイルはオペレーティングシステムのコンフィギュレーションを XML 形式で記述したものです。

- `*.c` および `*.h`

  プロジェクトとそのコンポーネントの C コードファイルおよびヘッダファイルです。`*.six` ファイルの以下のブロック内に、INTECRIO が使用する `*.c` ファイルと `*.h` ファイルが記述されます。

```
<fileContainer complete="false">
     <pathBase path="{{codeDir}}" />
     <!-- model specific C files -->
     ... *.c- and *.h files ...
</fileContainer>
```

上記のファイル以外にもいくつかのファイルが生成されますが、これらのファイルは INTECRIO での作業には関係ありません。

転送が完了すると、INTECRIO でプロジェクトの実験を行うことができますが、転送時の設定内容に応じて、実験を開始する際の操作が異なります。

実験を開始する：

以下の手順の個々のステップの具体的な操作方法は、INTECRIO のマニュアルに説明されています。

- マニュアル操作でINTECRIOにコードをインポートします。

  このステップは、コードが自動的にインポートされた場合は不要です。

- モデルをINTECRIOシステムプロジェクトに追加します。

  このステップは、コードが自動的に統合された場合は不要です。
- システムプロジェクトを完成させます。

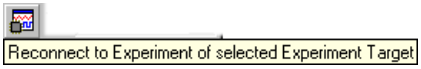
  このステップには、ハードウェアシステムの作成、オペレーティングシステムの設定、およびハードウェアとソフトウェアの接続が含まれます。
- オペレーティングシステムの設定を、マニュアル操作、または `*.oil` ファイルを読み込むことによって行います。
- 実行ファイルを生成します。
- 実験を開始します。

INTECRIO の実験環境の使用方法は、『INTECRIO ユーザーズガイド』に説明されています。

ASCET で INTECRIO のバックアニメーションを行う際は、特殊な実験環境を使用します。そこでは通常の方法でパラメータの適合を行うことができます。測定システムも通常どおりに動作しますが、ASCET のオフライン／オンライン実験とは異なり、オシロスコープ、レコーダ、およびデータロガーの機能は使用できません。INTECRIO でのバックアニメーションでは、これらの機能に必要な同期測定が行えないためです。その代わり、INTECRIO 上で別の測定インストゥルメントを使用できます。

バックアニメーションを行う：

- INTECRIO で、プロジェクトの実験を開始します。
- ASCET プロジェクトエディタで、“Experiment Target” に `INTECRIO` が選択されていることを確認してください。
- **Reconnect to Experiment of Selected Experiment Target** ボタンをクリックします。



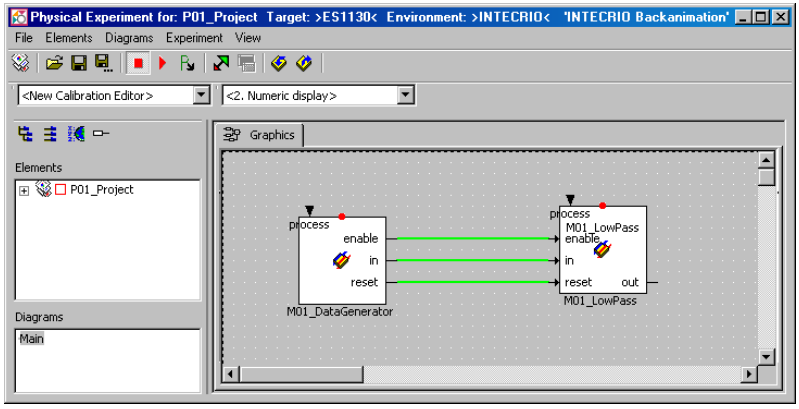
または

- **Component → Reconnect to Experiment** を選択します。

  ハードウェアオプションの **Use ETAS Network Manager (enables ‘Select Hardware’)** オプション（2.1 項を参照）がオンになっていると、状況に応じてハードウェア選択ダイアログボックス（2.2.1 項を参照）が開きます。

このダイアログボックスの使用方法は、36 ページ「ハードウェアを選択する（ETAS ネットワークマネージャを使用）：」に説明されています。

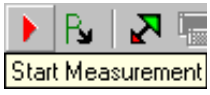
INTECRIO で実験を実行するための接続が確立され、“Physical Experiment...”ウィンドウが開きます。タイトルバーに“INTECRIO Backanimation”と表示され、バックアニメーション専用の実験環境であることがわかります。



通常のオフライン／オンライン実験とは異なり、ウィンドウには“Graphics”タブのみが表示されます。

- 必要な測定ウィンドウを作成して設定します（『ASCET ユーザーズガイド』の「測定システム」の項を参照してください）。
- 必要な適合ウィンドウを作成して設定します（『ASCET ユーザーズガイド』の「適合システム」の項を参照してください）。
- **Experiment → Start Measuremet** を選択します。

または



- **Start Measurement** ボタンをクリックして測定を開始します。

  測定／適合ウィンドウの表示内容が周期的に更新されます。

バックアニメーション用の実験が開いている場合、NTECRIO の実験で使用されているハードウェアがチェックされ、実験が ES1135 または ES910 上で実行されている場合は、NVRAM コックピット（73 ページの「NVRAM コックピット」を参照してください）が使用可能になります。

実験のロード、保存、エクスポートの方法は、『ASCET ユーザーズガイド』の「環境設定のロードと保存」の項に説明されています。ロードされた環境内に、バックアニメーションで使用できないエレメント（オシロスコープなど）が含まれていると、それらのエレメントは無視されます。

ここでは、モニタ機能（『ASCET ユーザーズガイド』の「モニタ機能」の項を参照してください）を使用して、数値変数と論理変数をモニタすることもできます。モニタ機能の設定は、環境内に保存されます。

また、プロジェクト内のコンポーネント間をナビゲートすることができます（『ASCET ユーザーズガイド』の「ブロックダイアグラム間のナビゲート」の項を参照してください）。

プロジェクトにステートマシンが含まれている場合、ステートマシンのアニメーション機能（『ASCET ユーザーズガイド』の「ステートマシンの検証」の項を参照してください）を利用できます。

実験で得られたデータを ASCET モデルやハードディスクに書き込むことができます。またハードディスクからデータを読み込むことも可能です。詳しくは『ASCET ユーザーズガイド』の「データの扱い」の項を参照してください。

バックアニメーションを終了する：

- **File → Exit** を選択します。

または

- **Exit to Component** ボタンをクリックします。

  バックアニメーションが終了し、実験環境が閉じます。INTECRIO の実験はそのまま続行されます。

5 RTIO パッケージ

5.1 概要

Real**T**ime **I**nput/**O**utput Package（RTIO パッケージ）は、ASCET と ETAS 実験ハードウェアシステム間の接続を容易に実現するためのものです。VME バスベースの ETAS 実験システムハードウェアは、さまざまな目的に使用できる開発／実験用プラットフォームです。アナログ信号やデジタル信号の入出力には、ES シリーズの VME バスインターフェースボードを使用します。

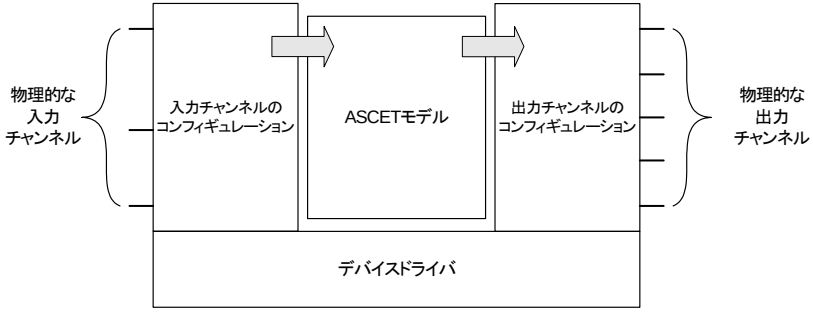
この RTIO パッケージのユーザーインターフェースを利用して、複数のハードウェアコンポーネント（割り込みにより制御されるものも含みます）の ASCET プロジェクトへの組み込みとコンフィギュレーション設定を容易に行うことができます。

組み込まれたハードウェアについては、ターゲットシステム専用の C コードが自動生成されます。

5.2 RTIO パッケージのアーキテクチャ

ASCET-RP を使用すれば、ASCET モデルから、ES1000 システムにおいてリアルタイム環境で動作するプロトタイプソフトウェアを容易に生成することができます。

このためには、シミュレーションボードとその他何枚かの I/O ボードで構成された ES1000 システムのハードウェア環境に、ASCET モデルを組み込む必要があります。この際、ASCET モデルは論理レベルで I/O チャンネルに接続されます。



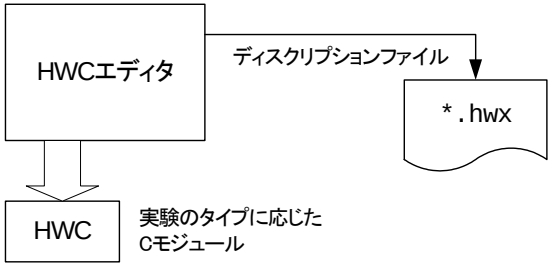
この接続を行う作業は、2 つのステップで行います。HWC エディタ（第 7 章を参照してください）を使用して、まず I/O ボードのコンフィギュレーション、つまり物理的な入出力チャンネルの設定を行い、次にそれらのチャンネルを ASCET モデルの入出力である ASCET メッセージに割り当てます。

物理チャンネルの設定（ハードウェアコンフィギュレーション）は、XML ディスクリプションファイル（*.hwx）に書き込まれ、同時に ASCET プロジェクト内にも保存されます。

注記

デフォルト状態において、ハードウェアコンフィギュレーションは XML フォーマット（*.hwx）で保存されます。
必要に応じて既存の *.hwc フォーマットを選択して使用することもできますが、ASCET-RP の将来のバージョンでは *.hwc フォーマットはサポートされなくなるため、できる限り XML フォーマットをお使いいただくことをお勧めします。

プロトタイプソフトウェアを生成するため、HWC エディタには、選択された実験のタイプ（物理実験、量子化実験、実装実験）に応じたターゲット用の C コード（第 8 章を参照してください）を生成する機能もあります。生成された C コードは、各プロジェクト用に生成される HWC（5.2.1 項を参照してください）という C モジュールに保存され、このモジュールは、ASCET データベースに格納されます。



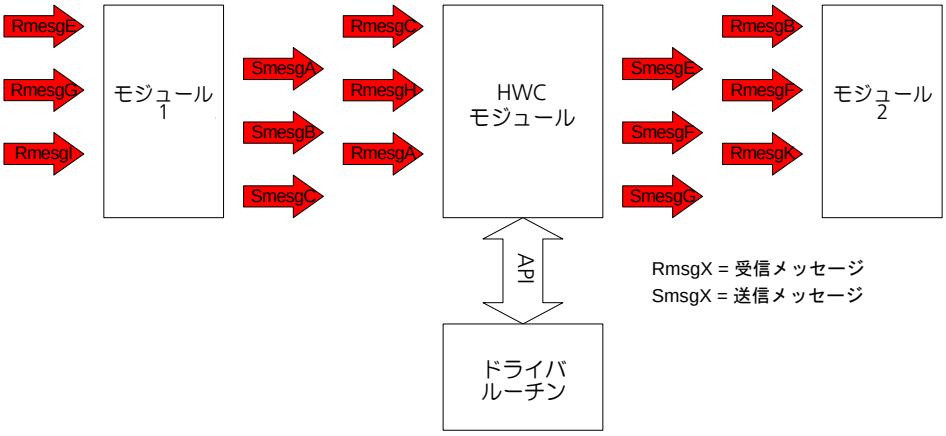
ASCET モデル全体のコードが生成されてコンパイルされる際、ハードウェアコンフィギュレーションの C コードモジュール（「HWC モジュール」）も同時にコンパイルされて I/O ドライバにリンクされ、実験ターゲット上で実行可能なプロトタイプが生成されます。

5.2.1 HWC モジュール

RTIO パッケージを使用する ASCET プロジェクトには、上述の「HWC モジュール」が必要です。これは、ハードウェアとの接続に必要な C コードを格納するためのコンテナとして使用される C コードモジュールです。RTIO フレームワークは、この C コードモジュールを“HWC”というインスタンス名により識別します。

HWC モジュールは、メッセージを使用してモデル内の他の部分と通信します。この方法によって、モデル内のハードウェア制御部分とそれ以外の部分とが明確に分離されます。

下図は、RTIO プロジェクト内のモジュール間の関係を示しています。



HWC モジュール用の C コードは、各システムやボードの専用ドライバルーチンの API[1] インターフェースに合わせて調整されるので、システムは、ドライバルーチンをライブラリとして使用することが可能となります。HWC モジュール内の API 関数呼び出しやドライバ用データは、各ドライバに依存しない標準の記述形式で定義されています。

このようにして、新しいボードを容易に統合できるモジュラーシステムを構築できます。

5.2.2 HWC エディタ

ハードウェアコンフィギュレーションエディタ（HWC エディタ）は、RTIO フレームワークの中核となる機能です。

HWC エディタは、プロジェクトから起動され、ここで、使用される実験システムに対応するハードウェアコンフィギュレーションを設定します。そして、コード生成時には、ハードウェアコンフィギュレーションに組み込まれたハードウェアに対して必要なエレメントとその C コードが生成され、ハードウェアにアクセスするための環境が実装されます。

1. API = Application Programming Interface

6 準備

本項では、RTIO パッケージをインストールして使用するための一般的な条件と準備作業ついて説明します。

6.1 ハードウェア - ES1000.x 実験システム

RTIO パッケージを稼働させるためには、いずれかの ETAS 実験ハードウェアシステム（VME システム）が必要です。通常は以下のような機種が使用され、これらを組み合わせて使用することもできます。

- ES1000.2
- ES1000.3

RTIO フレームワークは、ETAS のシステムコントローラボードをすべてサポートしています。標準構成例を下図に示します。

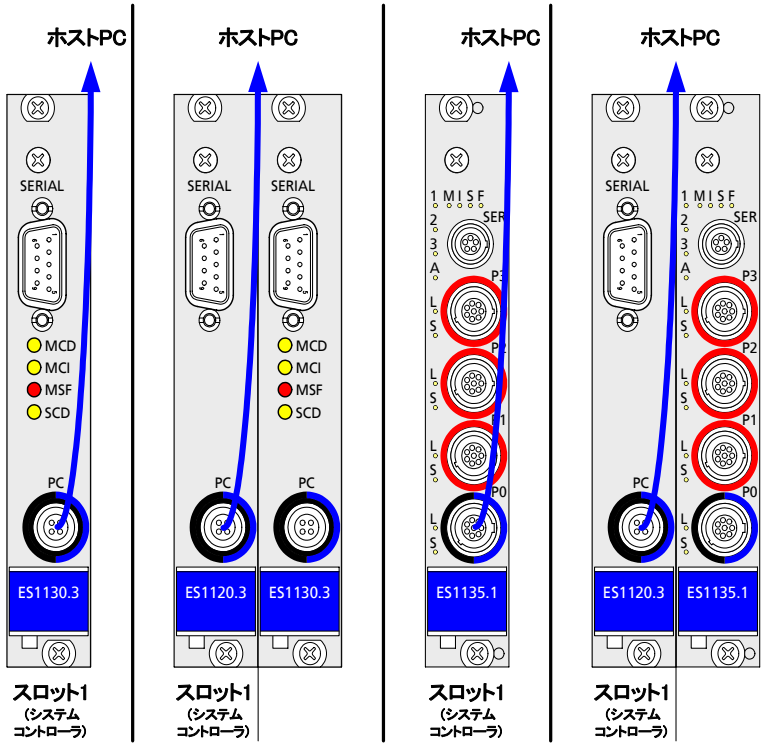


図 6-1 標準的なシステムコントローラ構成

コントローラボード ES1120 とシミュレーションボード ES1130 / ES1135

ES1000.x を使用してラピッドプロトタイピングと適合作業を並行して行う場合は、ホスト PC との通信は、コントローラボード ES1120 を介したイーサネット接続によって行われます。ASCET で開発されたファンクションを、コントローラボード ES1120 経由で PPC モジュール ES1130 または ES1135 にロードして実行し、この環境において、プロトタイピング実験とデータ測定／適合を並行して行うことができます。

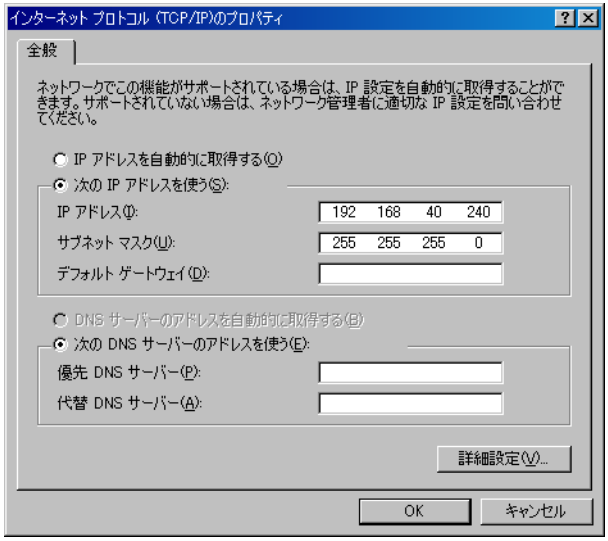
TCP/IP プロトコルオプション

社内 LAN 用に別のネットワークカードを使用している場合、IP アドレスの競合を防ぐため、TCP/IP を以下のように設定する必要があります。

TCP/IP プロトコルオプションを設定する：

- DHCP サービスを無効にします。
- IPアドレスフィールドに `192.168.40.240` と入力します。
- サブネットマスクフィールドに `255.255.255.0` と入力します。

（Windows® 2000 の場合）



- DNS サービスを利用する場合は、社内 LAN のローカル設定を使用してください。
- WINS サービスを無効にします。
- IP 転送オプションが無効になっていることを確認します。

6.2 ES1135 の特殊な機能の紹介

ES1130 の後継機種である ES1135 には、いくつかの機能が追加されています。ここではこれらの機能について説明します。

6.2.1 不揮発性 RAM（NVRAM）

基本事項

「不揮発性（NV）変数」つまり NVRAM にマッピングされる適合変数は、ASCET で記述される他の適合変数と同様に使用できます。この変数に対しては、モデルからの読み書きのほか、適合ウィンドウでの適合処理や、データ測定やロギングを行うことができます。NV 変数の特徴は、シミュレーションを中断した場合、その後、同じモデルを用いるシミュレーションを再開するときに、中断時の値をそのまま継続して使用できるという点です。これは、ECU コード内の自己学習アルゴリズムに使用されるアダプティブ適合変数や、診断結果を保存する変数などに利用できます。

NV 変数であることを表わす *non-volatile*（NV）という属性は、ASCET のすべての基本データタイプ（スカラ、配列、マトリックス、特性カーブ／マップ）の適合変数に対して任意に割り当てることができます。ただし、NV 変数として使用できるのはこれらの ASCET 変数だけで、C コードで記述された変数を NV 変数として使用することはできません。

NV 変数を定義するには、クラスエディタまたはモジュールエディタにおいて、変数のエレメントエディタ上の **Non-volatile** オプションをオンにします。

注記

NVRAM を使用するプロジェクトには、**ユーザー定義**の INIT プロセスを用意し、すべての NV 変数の値の妥当性を、各変数ごと、および変数間でチェックする必要があります。このチェックが確実に行われない場合は、すべての NV 変数を所定のデフォルト値（安全な値）に初期化しなければなりません。
これは、NVRAM 上の不正な値が初期値として使用されたことによって人体や機材を傷付ける可能性が生じるような環境（車両やテストベンチなど）においてプロジェクトを稼動させる場合は、**絶対に必要**な処置です。

ハードウェアの特徴

ES1135 のメインプロセッサである IBM750GX のアドレス空間には、64K バイトの NVRAM（不揮発性 RAM）が含まれています。このメモリ領域に格納された値は、電源障害の復旧後、または電源を一度切って再投入した後でも、そのまま保たれています。

NVRAM へのアクセスは、通常の揮発性 RAM に比べてはるかに時間がかかる上、キャッシングも行えません。そのため、パフォーマンス上の理由から、NV 変数は NVRAM に直接マッピングされるのではなく、通常の揮発性変数と同様に通常の

RAM にマッピングされ、その値が定期的に NVRAM に保存されます（自動更新モード）。デフォルトの更新周期は 10 秒ですが、以下の API 関数（13.3 項を参照してくださいを参照してください）で、更新周期を 1 ～ 30 秒の範囲内で任意に設定することもできます。

```
uint32 nvramSetUpdateInterval(uint32 interval_sec)
```

NVRAM への値の保存はアイドルタスク内で行われるので、モデルのリアルタイム挙動には影響しません。

NVRAM は、データの整合性を保つために交代バッファ方式になっているので、実際に使用できる容量は、メモリサイズの半分です。また、多少のオーバーヘッドもあるため、最終的に ASCET モデルが使用できる NVRAM 容量は、32K バイト弱となります。

ES1135 のファームウェアでは、この容量制限が考慮されています。モデル内に定義された NV 変数の合計サイズが NVRAM の容量を超えていると、実験開始時に“NVRAM overflow”というエラーメッセージが ASCET モニタウィンドウに表示されます。この場合、NVRAM はまったく使用されません。つまり、値は NVRAM に保存されません。このような場合は、モデル内の NV 変数の数を減らしたり、サイズを小さくする必要があります。

NVRAM データにはアドレス情報や変数名は含まれないため、それらのデータは、前回の更新後にモデル内の NV 変数の構成が変わっていない場合にのみ、モデルに適用されます。このために、コード生成時に特殊な NV 識別子が作成され、その変更の有無がチェックされます。

NV 識別子は、以下のアクションによって変更されます。

- プロジェクト、モジュール、クラス、サブクラス内のいずれかの NV エレメントのインスタント名の変更
- プロジェクト、モジュール、クラス、サブクラス内のいずれかの NV エレメントのインプリメンテーションの変更

  これは、以下の条件を指します。

  - コード生成オプションが `Physical Experiment` の状態で、実装データ型を `cont` から `sdisc/udisc` に変更したとき
  - コード生成オプションが `Implementation Experiment` の状態で、実装値のインターバル（範囲）を変更したとき
- いずれかの NV 変数の変換式パラメータの変更

  変換式の名前やコメントのみを変更した場合、NV 識別子は変更されません。
- NV 列挙型データの場合：
  - 列挙子の順番の変更
  - 列挙子の名前（値）の変更
  - 列挙子の削除／追加
  - 異なる列挙型データを選択（含まれる列挙子が同じである場合を含む）

- 多次元の NV エレメント（配列、マトリックス、特性カーブ／マップ）の最大サイズの変更
- 実装実験（コード生成オプションが `Implementation Experiment`）における、実装値のインターバルの変更
- プロジェクト、モジュール、クラス、サブクラス内での NV エレメントの削除／追加
- ステートマシン内のステートの追加／削除

NV 識別子は、以下のアクションによっては変更されません。

- プロジェクト名の変更
- プロジェクト内のモジュールまたはクラスの名前の変更
- プロジェクト、モジュール、クラス、サブクラス内に含まれるいずれかの通常の変数（Volatile 変数）のインスタンス名の変更
- いずれかの NV エレメントの変換式の、名前またはコメントの変更

  変換式のパラメータが変更されたときは NV 識別子も変わります。
- 列挙型データの名前の変更
- NV エレメントと通常のエレメントの値の変更
- 多次元の NV エレメント（配列、マトリックス、特性カーブ／マップ）の実効サイズの変更
- NV エレメントおよび通常のエレメントに使用されるメモリ領域の変更

NV 変数の初期化と更新

シミュレーションの開始時： モデルコードがターゲットにダウンロードされた時、そのモデルに含まれる NV 変数について、NVRAM 内に対応するデータがない場合は、その NV 変数にデフォルト値が初期設定されます。「対応するデータがない」というのは、NV メモリが空であるか、チェックサムの照合によって不一致が認められたか、または NV 識別子のチェックにより NV データとダウンロードされたモデルとの不整合が認められた、ということです。

一方、対応するデータが NV メモリ内に保存されていた場合には、その値がモデルの初期値として使用され、ERCOS^{EK} を起動して実験を開始できるようになります。

シミュレーションの終了時： ERCOS^{EK} を停止してシミュレーションを終了すると、最後に保存された NV 変数の値が NVRAM 内に保存されます。ターゲットの電源がオフになったり、コードが再度ダウンロードされた場合でも、最後に保存された NV 変数値を用いてシミュレーションを再開できます。

前述のように、NV 変数の値は自動更新モードでは NVRAM に定期的に保存されますが、値を確実に NVRAM に格納するためには、以下の API 関数を、Exit タスク（アプリケーションモードが `inactive` であるタスク）の最後に呼び出してください。

```
void nvramUpdateMemoryExit(void)
```

現在使用されているモデル内に NV 変数が 1 つもない場合には、NVRAM の内容はまったく変更されません。

NV 変数を含むモデルがフラッシュメモリ内に書き込まれている場合： シミュレーションコントローラの FLASH メモリ内に NV 変数を使用するシミュレーションモデルが書き込まれている場合、このモデルは電源の投入によって通常どおりブートされますが、シミュレーションが開始される前に対応する NV データが保存されているかどうかが調べられ、あればその値が初期値として書込まれます。

モデルがデフォルトの NV 変数値に基づいて実行されているかどうかを調べる： モデルの初期化時において、ASCET のデータエディタで定義されたデフォルト値が NV 変数に設定されたのか、あるいは NVRAM に保存されていた値が設定されたのかは、実験環境の Target Debugger ウィンドウに表示される情報メッセージで判断できます。また、モデル内から以下の API 関数を実行しても、このメッセージと同じ情報が得られます。

```
uint8 nvramCheckForInitializedVars(void)
```

NVRAM の内容をクリアする： 前記のとおり、プログラム識別子が整合している場合は NVRAM の値が NV 変数の初期値として使用されますが、必要に応じて NVRAM の内容を任意にクリアし、NV 変数にモデルのデフォルト値が初期値として設定されるようすることができます。現時点において、この機能は GUI ではサポートされていませんが、以下の API 関数を使って、モデルから NVRAM をクリアすることができます（13.3 項 「API 関数（NVRAM）」を参照してください）。

```
uint32 nvramClear(void)
```

データの整合性

電源断またはシステムクラッシュによりシミュレーションが中断された場合でも、NV 変数の値は NVRAM に格納されています。しかしこれらの値が妥当なものであるかどうかは、自動更新またはモデル内からの操作によって最後に値が保存されたときのタイミングによります。

このため、予想外の中断が発生した時の NVRAM の内容の整合性を、以下に示すようなさまざまなレベル、または方法で確保することができます。

以下の API 関数では、3 通りの整合性レベル（なし、低レベル、高レベル）を設定します。

```
uint32 nvramSetConsistencyLevel(T_consistencyLevel
level)
```

整合性なし： NVRAM の更新は、すべてのタイプの NV 変数の整合性を無視して行われます。

低レベルの整合性（単一変数の整合性）： このレベルにおいては、スカラ、配列、およびマトリックスタイプの各 NV 変数単位でのデータ整合性は保証されますが、特性カーブ／マップの整合性は保証されません。

このレベルにおいては、特性カーブ／マップのような複合タイプの変数の更新をアトミックに、つまり中断されることなく行うことはできません。

高レベルの整合性（複合変数、およびタスク終了後の整合性）： このレベルにおいては、アイドルタスクにおいてすべての NV 変数の値が中断されることなくローカルバッファに保存された場合のみ、NVRAM の更新が行われます。

アプリケーションにおいて、一連の NV 変数の値を変更する処理は、アトミックに実行されるようにしてください。たとえば、自己学習アルゴリズムは複数の変数を対象に処理されるので、NVRAM 内の複数の変数の値が、同じ演算サイクルで得られるようにする必要があります。

モデルにより制御される整合性（複合変数、および複数のタスクサイクル間の整合性）： 上記の「高レベルの整合性」メカニズムは、NV 変数の操作が 1 つのタスクサイクル内で行われる場合に限り整合性を保証します。しかし、NV 変数の操作は複数のタスクサイクルにわたって行われる場合もあり（たとえば、1 つの適合変数を複数のタスクで更新する場合など）、そのような場合、ES1135 ファームウェアはそれを認識できないので、モデルの側で NVRAM の更新タイミングを制御する必要があります。

自動更新をオフにするには、以下の関数を用います。

```
uint32 nvramDisableAutoUpdate(void)
```

また、次の関数で、すべての NV 変数の更新をモデルから任意に開始することができます。

```
uint32 nvramManualUpdateBackground(void)
```

任意更新を行う場合でも、更新処理が完了するまでの間にシステム全体がブロックされたりリアルタイム挙動が変わってしまったりすることがないように、この関数は、呼び出された後にすぐに呼び出し元に戻り、実際の更新はバックグラウンドで行われます。この際、以下の関数を用いてバックグラウンドでの更新ステータスを調べることができます。

```
uint8 nvramCheckRunningUpdate(void)
```

この関数は、更新処理が実行中である場合には `true` を返します。ユーザー側（モデル）は、更新中に NV 変数が変更されないようにしなければなりません。

自動更新を再開するには、以下の関数を使用します。

```
uint32 nvramEnableAutoUpdate(void)
```

任意更新の際、一部の NV 変数だけを選択して更新することはできません。

現在のモデル内に NV 変数が 1 つも含まれていない場合には、NVRAM の内容はまったく変更されません。

NVRAM の内容の不整合： NVRAM の内容に不整合があった場合、たとえば、チェックサムの照合結果が不良だった場合には、実験環境、または ASCET モニタウィンドウに、警告メッセージが出力されます。

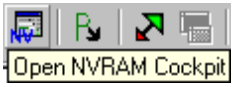
NVRAM コックピット

NVRAM 用 API（13.3 項を参照してください）には NVRAM の更新を制御するための機能が含まれています。実験中に、「NVRAM コックピット」という専用ウィンドウを使用してこの機能を利用することができます。このウィンドウには、API 関数を使用して設定された内容も反映されます。

NVRAM コックピットを使用する：

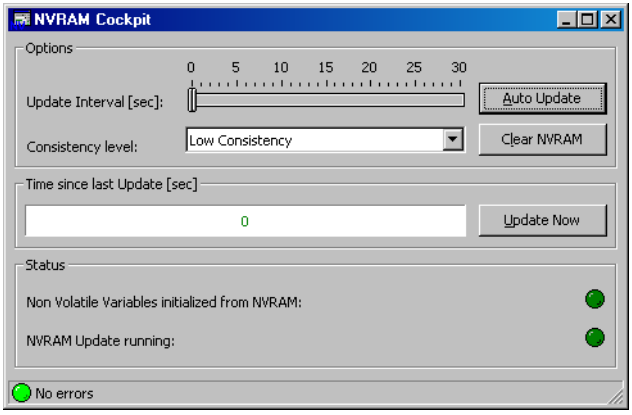
- 実験ウィンドウから **Experiment → NVRAM Cockpit を選択します。**

または



- **Open NVRAM Cockpit** ボタンをクリックします。

  NVRAM コックピットが開きます。



- 必要に応じて各コントロールエレメントを使用します。
- 操作が終了したら、**Close** をクリックして NVRAM コックピットを閉じます。

NVRAM コックピットには、以下のコントロールエレメントが含まれています。

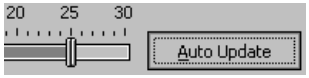
- **Update Interval [sec]（更新周期の設定）**

  このスライダエレメントで、NVRAM の内容の自動更新の周期（秒単位）を調整します。デフォルトは 10 秒で、30 秒まで設定可能です。

  スライダは、自動更新が ON になっている場合のみ使用可能です。

- **Auto Update （自動更新 ON/OFF）**

  このボタンで、自動更新の ON/OFF を切り替えます。自動更新は、このボタンが押された状態（a）になっていると行われ、押されていない状態（b）においては行われません。.

  (a) 

  (b) 

注記

NVRAM の内容の自動更新は、実験の実行中にのみ行われます。
Experiment → Stop ERCOS または **Stop ERCOS** ボタンで実験を終了すると、NVRAM の内容のは自動更新されなくなります。

- **Consistency level （整合性レベルの設定）**

  このコンボボックスで、更新の整合性レベルを選択します。詳しくは 72 ページの「データの整合性」を参照してください。

- **Clear NVRAM （NVRAM の消去）**

  このボタンで NVRAM の内容を消去できます。

  自動更新が有効になっているときに **Clear NVRAM** をクリックすると、NVRAM の内容が消去されますが、次回の自動更新のタイミングにおいて再度書き込みが行われます。

- **Update Now（任意更新）**

  このボタンをクリックすると、NVRAM の内容が更新されます。

  このボタンは、自動更新が OFF になっている場合にのみ使用できます。

注記

NVRAM の内容の任意更新は、**Experiment → Stop ERCOS** または **Stop ERCOS** ボタンで実験を終了した後も、任意に行えます。

NVRAM コックピットには、以下の表示エレメントが含まれています。

- **Time since last Update [sec]（前回の更新からの経過時間表示）**

  この棒グラフに、前回更新が行われた時からの経過時間が表示されます。全体の幅は 30 秒を表します。もしも自動更新が OFF になっていて 30 秒以上更新が行われないと、秒数を表す数字だけが更新されます。

  時間のカウントは、実験終了後も継続されます。実験終了後に任意更新を行うと、カウンタがリセットされ、新たにカウントが開始されます。

  前回の更新から 30 秒が経過するまでは棒グラフは緑で表示され、それ以上経過すると赤で表示されます。ただし、30 秒経過する前に実験を終了（**Stop ERCOS**）した場合は、30 秒経過した時点で黄色表示になります。

- **Non Volatile Variables initialized from NVRAM（Non-Volatile 変数が NVRAM の値で初期化されたかどうかを表示）**

  NV 変数が NVRAM 上の値で初期化されると、明るい緑の LED（a）が表示されます。暗い緑（b）の LED は、NV 変数がデフォルト値で初期化されたことを示します。

  (a) 

  (b) 

- **NVRAM Update running（NVRAM の更新状態）**

  NVRAM の更新処理が実行されている間、明るい緑の LED が表示されます。

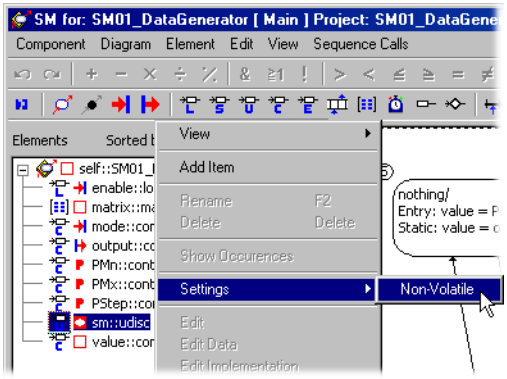
  

NVRAM 使用上のヒント

以下に、NVRAM を使用する際に役立つヒントを紹介します。

- ステート変数を NVRAM 変数として定義する

  NVRAM のサポート機能が強化され、ステートマシンのステート変数 sm に対して Non-Volatile 属性を割り当てることができるようになりました。そのためには、ステートマシンエディタの“Elements”リストの sm 変数を右クリックし、**Settings → Non-Volatile** を選択します。



- 集合エレメントの実際のサイズの保存

  Non-Volatile 性が割り当てられた集合エレメント（配列、マトリックス、特性カーブ／マップ）の実際のサイズ（“current size”属性）は、NVRAM に保存されます。

  オフライン時にモデル内のこれらの変数のサイズを変更し、プログラムをダウンロードした場合、新しいサイズが有効になるのは、NVRAM コックピットからの操作などによって NVRAM の内容が消去された後、または NVRAM の内容とプログラムの内容とが不一致になった後です。

- NVRAM を使用するプロジェクトを ES1000 のフラッシュメモリ上で実行する際の注意点

  フラッシュメモリ上に書き込まれたプログラムは、ES1000 の電源をオンにするたびに起動される、ということを常に念頭に置いておくようにしてください。このプログラムが NV 変数を使用する場合、その変数は NVRAM 上に配置されます。その状態で、NV 変数を使用する別のプログラムをフラッシュメモリに上書きすると、NVRAM の内容がリセットされてしまう可能性があります。

6.2.2 ウォッチドッグ

ラピッドプロトタイピングシステムの安全性を高めるため、ES1135 にはハードウェアウォッチドッグ機能が搭載されています。これは ES1135 のメインプロセッサをモニタする、独立した制御ユニットです。あらかじめ定義されているデータシーケンスがメモリセル（「ウォッチドッグサービスレジスタ」）に周期的に書き込まれ、もしもこの書き込みアクセス（「ウォッチドッグサービス」）が成功しないまま最大時間（「ウォッチドッグ周期」）が経過すると、プロセッサの例外処理（「イベント」）がトリガされます。

ES1135 HW ウォッチドッグは、以下の 2 つのモードで使用できます。

1. 安全性を優先するモード（「セーフティモード」）
2. 機能が強化されたフレキシブルモード（「RSEF（**R**educed **S**afety **M**ode **E**nhanced **F**unction）モード」）

RSEF モードにおいては、実行中のモデルから、ウォッチドッグ機能を以下のように再設定することができます。

- イベント設定

  ウォッチドッグタイマ満了時に実行される例外処理を定義します。ウォッチドッグがオフになるように設定することも可能です。

- ウォッチドッグ周期

  ウォッチドッグサービス（値の書き込み）の発生の有無を監視するウォッチドッグタイマのリミット値を定義します。

- モード切り替え

  ウォッチドッグ周期とイベントが適切に設定されていないと、セーフティーモードに切り替わり、その後は、モードを再設定したり、別のモードに切り替えることはできなくなります。不正なイベントの発生を防ぐため、あらかじめ、ウォッチドッグサービスを適切に設定しておいてください。

電源電圧がオンになると、ウォッチドッグは RSEF モードになり、セーフティーモードはオフになります。

ウォッチドッグサービス

ウォッチドッグサービス（メモリセルへのデータ書き込み）がウォッチドッグ周期が満了する前に行われないと、所定のウォッチドッグイベントが発生します。モデルを設計する際は、必ず、モデルの誤動作を確実に検知できる位置でこのサービス関数を呼び出すようにしてください。

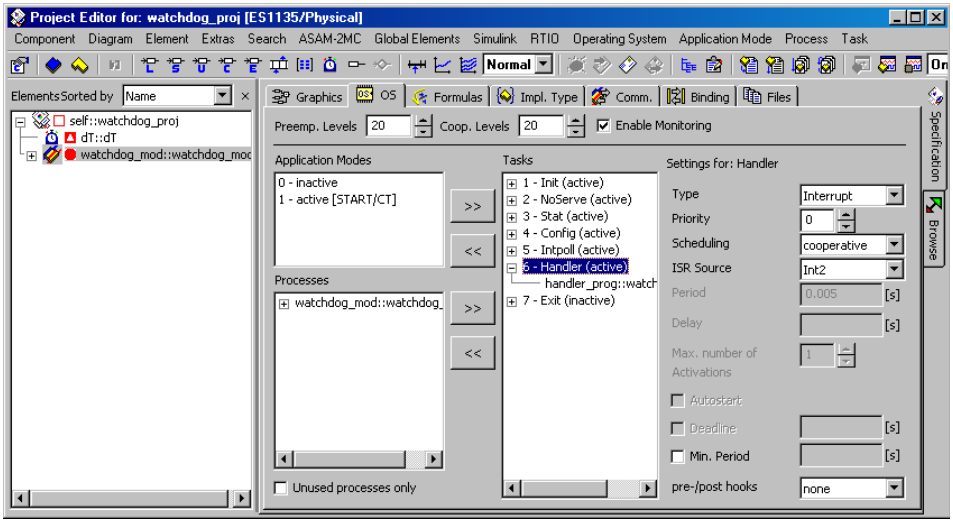
シミュレーションコントローラのファームウェアには自動ウォッチドッグサービスメカニズムが搭載されているので、モデルによって割込みが無効になっていなければ、30ms ごとにウォッチドッグサービスが行われます。従って、オペレーティングシステムが適切に稼動している限り、ウォッチドッグサービスは定期的に処理されます（ウォッチドッグ機能が有効になっている場合）。このメカニズムは、さまざまなケースに利用できます。

割込み制御

モニタやデバッグを行う際、ウォッチドッグタイマ満了時にシミュレーションプロセッサ割込みをトリガするように設定することができます。

割込みは、ポーリングしたり、また内部割込みコントローラに送ることもできます。ウォッチドッグ割込みはラッチされるので、明示的に応答（"acknowledging"）する必要があります。割込みソースのディセーブル／イネーブルの切替えは、専用の関数で高速に行うことができます。これらの関数の処理は、割込みの伝達にのみ影響します。

ウォッチドッグ割込みは、ASCET 内の HW タスクに割り当てられます。このタスクには、ASCET の OS エディタで、`Interrupt` というタスクタイプと `Int2` という ISR ソースを指定してください。



ウォッチドッグハンドラは、ERCOS^{EK} レベルよりも下、また他の HW 割込み（VME Bus からの割込みなど）よりも上のレベルで実行されるので、ウォッチドッグ割込みは、他の HW 割込みが処理されている途中であっても処理されます。割込みに対する応答は ES1135 ファームウェア内で行われるので、ハンドラタスク内では行わないようにしてください。ハンドラタスク内ではすべての ERCOS^{EK} 関数を使用できます。

ウォッチドッグ割込みが発生すると、ウォッチドッグは、250µs 後にタイムアウト状態から自動的に復元され、設定されている周期で新しい監視サイクルが再開されます。標準のウォッチドッグサービス `wdService()` を使用すれば、この復元処理を短時間で行うことができます。

ウォッチドッグ API については、13.4「API 関数（ウォッチドッグ）」という項で詳しく説明します。

6.2.3 LED

ES1135 のフロントパネル上の LED は、「システム LED」（M、I、S、F、A、L、S）と、自由にプログラミングできる「プログラマブル LED」（1、2、3）とに分かれています。

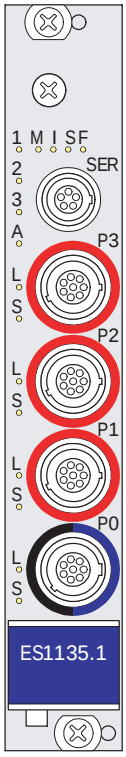


図 6-2 ES1135 - フロントパネル

システム LED については、ES1135 のハードウェアマニュアルに説明されています。

プログラマブル LED へのアクセスは、RTIO（10.2「ES1135-LED」を参照してください）、またはプログラムインターフェース（13.5「API 関数（ES1135 LED）」を参照してください）から行えます。

6.2.4 キャッシュロッキング

短い周期のタスクの実行時間は、そのタスクに該当するプログラムが、ES1135 のメインメモリに格納されているか、それともキャッシュ上に格納されているかにより、著しく変化します。処理タイミングが重要となるアプリケーションの場合、この違いがシステムに非常に大きな影響を与えてしまいます。

この対策として、ASCET-RP V5.5 では、モデル内で特に処理タイミングが重要となるパーツをマークし、そのパーツが常にキャッシュ上で実行されるように設定しておくことができます。この処理は「キャッシュロッキング」と呼ばれます。これにはセカンドレベルキャッシュ（「L2 キャッシュ」とも呼ばれます）が使用され、このキャッシュは 4 つのユニットに分かれています（「4 重キャッシュ」、「4 way キャッシュ」などとも呼ばれます）。

ここでマークできるのは、個々の変数やメソッド／プロセス全体です（81 ページ参照）。コンポーネントもマークすることができ、コンポーネントの設定は、82 ～ 85 ページの各設定方法に示された条件に基づいて、コンポーネントに内包されるエレメントにも適用されます。

キャッシュロッキングにおいては、以下の制限事項があります。

- キャッシュロッキングには、L2 キャッシュは最大 3 ユニットまでしか使用されません。これは、モデル内でマークされていないパーツ用に、最低 1 つのキャッシュが確保されている必要があるためです。
- キャッシュ内の配置により、キャッシュ上にスペースがあるにもかかわらず、マークされたすべてのモデルパーツを常にキャッシュに保持しておくことが不可能になる場合があります。このような場合、実験中にエラーメッセージが発行されます。

  このような状況はコードについてはほとんど発生することはありませんが、変数やパラメータについては頻繁に発生します。キャッシュロッキングはコードについて使用することをお奨めします。
- 4 つに分割されているキャッシュユニットをさらに細かく分割することはできません。たとえばキャッシュロッキング用にマークされたモデルパーツが 2.5 ユニットを必要とする場合、余りの 0.5 ユニットは使用されません。
- キャッシュの一部がキャッシュロッキング用に使用されている場合、マークされていないモデルパーツが使用できるキャッシュは少なくなり、その部分の実行遅延が発生する可能性があります。

モデルの各パーツに関して、キャッシュロッキングの設定を以下のモードから選択できます。

`Automatic`（デフォルト）	変数、パラメータ、メソッド、プロセスの場合、親コンポーネントの設定に合わせます。 コンポーネントの場合、`Automatic` は `Off` と同じ意味になります。.
`On`	キャッシュロッキング：オン
`Off`	キャッシュロッキング：オフ

変数、パラメータ、メソッド／プロセスのキャッシュロッキングを設定する：

- エレメントのインプリメンテーションエディタを開きます。

- “Cache Locking”コンボボックスで、モードを選択します。

  エレメント、メソッド、プロセスの設定は、親コンポーネントの設定に上書きされます。

注記

特性カーブ／マップの場合、1 つのタブ（“Value”、“X Distribution”、“Y Distribution“のいずれか）で選択されたモードが、すべてのタブに適用されます。

コンポーネントのキャッシュロッキングを設定する：

- コンポーネントのインプリメンテーションエディタを開きます。
- “Settings”タブを開きます。
- “Cache Locking”コンボボックスで、モードを選択します。

  選択されたモードは、`Automatic` モードに設定されているすべてのエレメント、メソッド、プロセスに適用されます。
  ただしこの設定はリカーシブルではないため、内包されるコンポーネントには適用されません。

内包されるコンポーネントのキャッシュロッキングを設定する：

注記

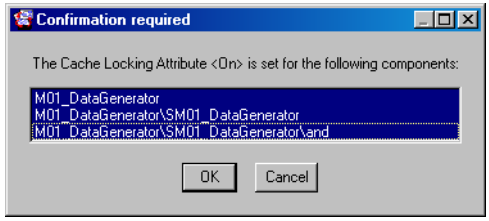
この設定はリカーシブルに適用されます。つまり内包されるコンポーネントに適用されます。

- プロジェクトまたはコンポーネントの“Elements”リストから、内包されるコンポーネントを選択します。
- **Elements → Set Cache Locking → *<setting>*** を選択します。

または

- 内包されるコンポーネントのショートカットメニューから **Set Cache Locking → *<setting>*** を選択します。

  確認のためのダイアログボックスが開き、選択されたコンポーネントとそれに内包されるコンポーネントがすべて表示されます。



- ***<setting>*** モードを割り当てたいコンポーネントを選択します。
- **OK** をクリックします。

  選択されたコンポーネントに ***<setting>*** モードが設定されます。コンポーネント内のすべてのエレメント、メソッド、プロセスには、`Automatic` モードが設定され、実際にはこのコンポーネントと同じモードが適用されます。

OS エディタ（プロジェクトエディタの“OS”タブ）では、以下のようにしてタスクを選択し、そのタスクに割り当てられたプロセスを含むモジュールについてキャッシュロッキングを設定することができます。

注記

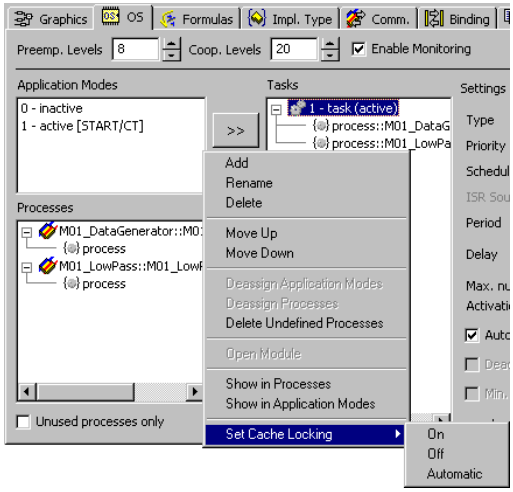
下記の方法では、**個々のプロセスのキャッシュロッキングを設定することはできません**。モジュール内の 1 つのプロセスのみが別のタスクに割り当てられていても、OS エディタではすべてのプロセスに同じモードしか設定できません。個々のプロセスについて設定する方法は、81 ページを参照してください。

OS エディタでキャッシュロッキングを設定する：

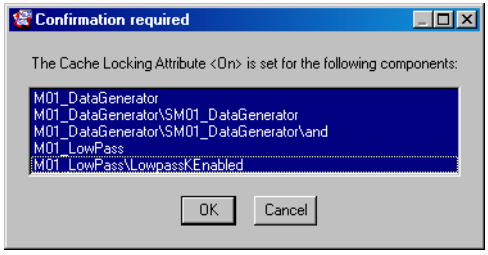
- OS エディタを開きます。
- “Task”ペーンで、タスクを 1 つ選択します。
- **Task → Set Cache Locking → *<setting>*** を選択します。

または

- タスクのショートカットメニューから **Set Cache Locking → *<setting>*** を選択します。



確認のためのダイアログボックスが開き、選択されたタスクに割り当てられたプロセスの親コンポーネントと、そこに含まれるコンポーネントがすべて表示されます。



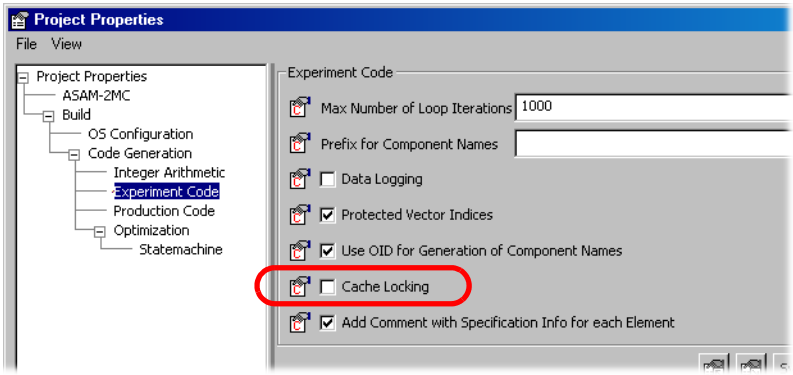
- ***<setting>*** モードを割り当てたいコンポーネントを選択します。
- **OK** をクリックします。

  選択されたコンポーネントに ***<setting>*** モードが設定されます。コンポーネント内のすべてのエレメント、メソッド、プロセスには、`Automatic` モードが設定され、実際にはこのコンポーネントと同じモードが適用されます。

以下の方法で、プロジェクトプロパティウィンドウの“Experiment Code”ノードで、プロジェクト全体のキャッシュロッキングのオン／オフを設定することができます。

プロジェクト全体のキャッシュロッキングを設定する：

- プロジェクトエディタで **Project Properties** をクリックして “Project Properties”ウィンドウを開きます。
- プロジェクト全体のキャッシュロッキングをオンにするには、“Experiment Code”ノードで **Cache Locking** オプションをオンにします。
- プロジェクト全体のキャッシュロッキングをオフにするには、上記オプションをオフにします。



プロジェクト内のすべてのエレメント、メソッド、プロセスに `Automatic` モードが設定され、実際にはここで設定したモードが適用されます。

6.3 システムソフトウェア

本書で説明する RTIO パッケージは、以下のソフトウェア製品の一部です。

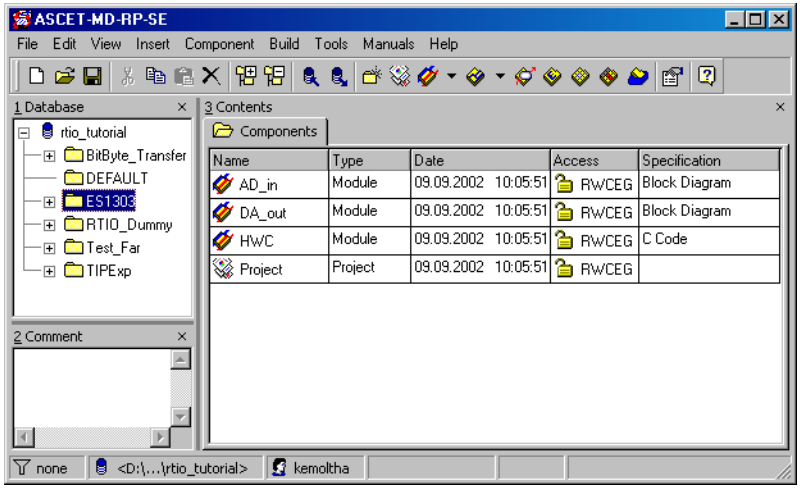
- ASCET（V5.1.1 以降）とそのアドオン製品 - ASCET-RP（V5.5.0 以降）

6.3.1 システムルートパス

ASCET-RP をインストールすると、ASCET ディレクトリの下に `Ascet5.2¥target` という新しいサブディレクトリが作成されるので、このサブディレクトリをシステムルートパスとして設定します。

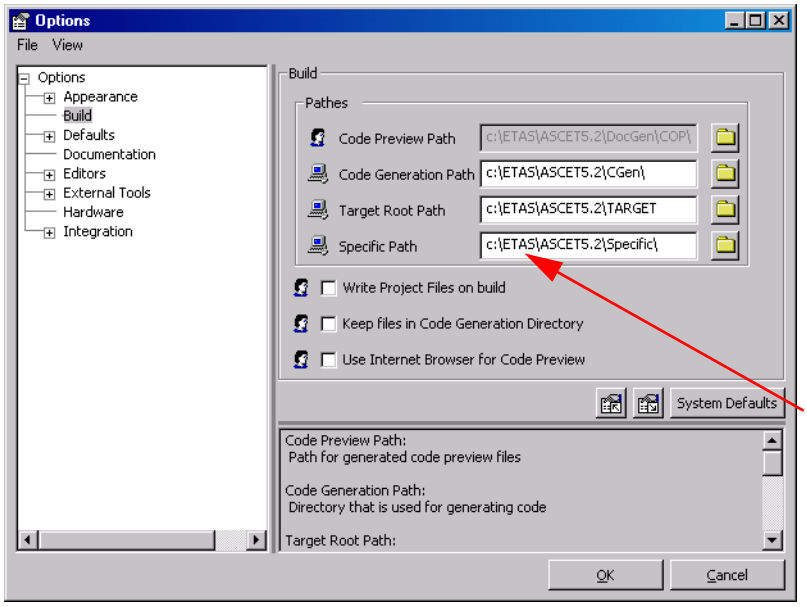
システムルートパスを指定する：

- システムルートパスは、ASCET の“Station Options”ダイアログボックスで設定します。



- ASCET のコンポーネントマネージャから、**Tools → Options** を選択します。

“Options”ウィンドウが開きます。



- “System Root Path”フィールドに **¥TARGET** というパスが指定されている必要があります。

6.3.2 C コードモジュール

RTIO によるハードウェア接続を行なうには、ASCET-SD プロジェクトごとに独立した C コードモジュールがデータベース内に生成される必要があります。

注記

従来のバージョンの RTIO には、データベース内のモジュール名についての制約（“HWC”）がありましたが、現在ではこの制約はなくなり、どのような名前でも使用できるようになっています。

6.3.3 プロジェクト

RTIO ハードウェアリンクを実行するためには、考慮しなければならない事柄がいくつかあります。

- RTIO コード生成を行なうプロジェクトには、プロジェクト専用に生成された C コードモジュールが追加されます。（前項を参照してください）。このモジュールには、“HWC”というインスタンス名が含まれていなければなりません。

- 使用するシステムコントローラのタイプ（ES1130 または ES1135）に応じて、ターゲットタイプを選択する必要があります。
- RTIO 接続をできるだけ容易に行うことができるように、プロジェクトエディタの“OS”タブのタスクリストに以下のタスクを定義してください。

タスク名	タスクタイプ	アプリケーションモード	機能
Init	Init	active	RTIO フレームワークがハードウェアドライバを初期化します。
Config	Software	active	一部のハードウェアドライバの再設定を行ないます。
Exit	Init	inactive	ドライバリソースを解放します。

  ハードウェアコンポーネントによっては、これ以外のタスクも必要です。これについては、各コンポーネントについての章で詳しく説明します。

- RTIO HWC モジュールがプロジェクト内の他のコンポーネント（他のモジュール、グローバルメッセージ）との通信に使用するすべてのメッセージは、通信相手側で“Exported”として定義されている必要があります。送信メッセージや送受信メッセージは、通常そのようなかたちで生成されますが、受信メッセージについてはそれが明示的に行なわれていることを確認してください。

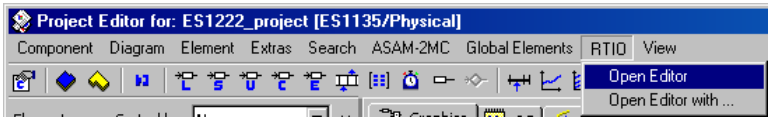
  従来のバージョンの RTIO とは異なり、階層モジュール内で生成されたメッセージにもアクセスできます。

7 HWC エディタ

HWC エディタは RTIO フレームワークの中核となる部分で、ハードウェアの構成と設定が定義された「ハードウェアコンフィギュレーション」を編集するためのエディタです。

7.1 HWC エディタを開く

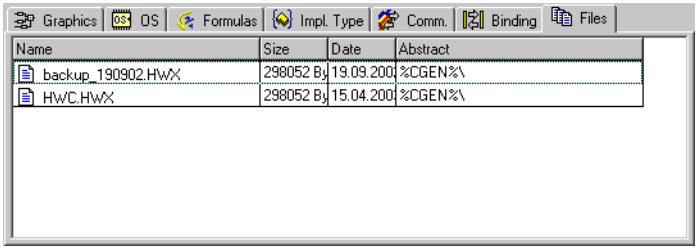
まずプロジェクトエディタで ASCET プロジェクトを開きます。RTIO パッケージが正しくインストールされていれば、メインメニューバーに **RTIO** メニューが表示されます。このメニューから **Open Editor** または **Open Editor with...** コマンドを選択すると、HWC エディタが開きます。



ASCET セッションにおいて HWC エディタを初めて開くときには、いくつかのシステムコンポーネントがロードされるため、時間が多少長めにかかります。（システムの起動を高速化してリソースを節約するため、一部の拡張機能は実際に必要になった時点でロードされます。）

パッケージが正しくインストールされている状態で、メインメニューバーの **RTIO → Open Editor** コマンドで HWC エディタを 開くと、ASCET プロジェクトの “Files” タブ（ファイルコンテナ）内で、`HWC.HWX`（または `HWC.HWC`）というファイルが検索されます。

ファイルが見つかると、それが自動的にエディタにロードされます。`HWC.HWX`（または `HWC.HWC`）ファイルの内容は、その HWC モジュールのコード生成が最後に実行されたときのものです。



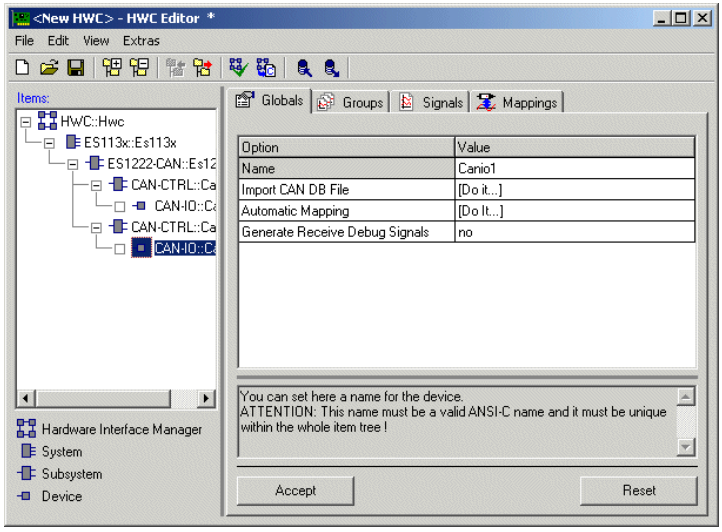
Open Editor with... コマンドで HWC エディタを 開く場合は、`HWC.HWX`（または `HWC.HWC`）以外の任意のハードウェアコンフィギュレーションファイルを選択してロードすることができます。

ハードウェアコンフィギュレーションファイルのファイルフォーマットの選択（`*.HWX` または `*.HWC`）は、“HWC Editor Options” ダイアログボックス（メニューコマンド **Extras → Options**）の “Files” タブで行えます。

ファイルを指定して開く場合は、プロジェクトエディタの“Files”タブでファイルを選択し、メニューコマンド **Project Files → Edit** を選択するか、またはショートカットメニューから **Edit** を選択します。

7.2 エディタの画面構成と操作方法

本項では、HWC エディタウィンドウの構成と一般的な操作方法について説明します。



ユーザーインターフェースは、ツールバー、“Items”リスト、および各オプション項目の値を入力したり修正するための一連のタブで構成されています。

タブの下のテキストフィールドには、選択された行についての簡単な説明文が表示されます。**Accept** ボタンをクリックすると設定の変更内容が有効になり、**Reset** ボタンをクリックすると変更内容は破棄されます。

7.2.1 ツールバー

本項では、ツールバーのボタンを使用して実行できる機能について説明します。

注記

個々の機能についての詳細は、95 ページの「メインメニュー」という項を参照してください。

新しいハードウェアコンフィギュレーションを作成する：

- 新しいハードウェアコンフィギュレーションを作成するには、**New** ボタンをクリックします。

または

- **File → New** を選択します。

ハードウェアコンフィギュレーションを開く：

- ハードウェアコンフィギュレーションを開くには、**Open** ボタンをクリックします。

または

- **File → Open** を選択します。

ハードウェアコンフィギュレーションを保存する：

- ハードウェアコンフィギュレーションを保存するには、**Save** ボタンをクリックします。

または

- **File → Save** を選択します。
- ハードウェアコンフィギュレーションを任意の名前で保存するには、**File → Save As** を選択します。

すべてのアイテムを展開して表示する：

- “Items”リスト内の枝をすべて展開して表示するには、**Expand all** ボタンをクリックします。

または

- **View → Expand all** を選択します。

すべてのアイテムを省略して表示する：

- “Items”リスト内の枝を可能な限り閉じて表示するには、**Collapse all** ボタンをクリックします。

または

- **View → Collapse all** を選択します。

ハードウェアを検索する：

- 現在接続されているハードウェアを検索するには、**Search for Boards** ボタンをクリックします。

または

- **Extras → Search for Boards** を選択します。

新しいアイテムを追加する：

- アイテムのリストに新しいアイテムを追加するには、**Add Item** ボタンをクリックします。

または

- **Edit → Add Item** を選択します。

アイテムを削除する：

- アイテムを削除するには、そのアイテムをリストから選択し、**Delete Item** ボタンをクリックします。

または

- **Edit → Delete Item** を選択します。

ハードウェアコンフィギュレーションをチェックする：

- ハードウェアコンフィギュレーションの妥当性をチェックするには、**Check Hardware Configuration** ボタンをクリックします。

または

- **Extra → Check Hardware Configuration** を選択します。

コード生成を開始する：

- コード生成を開始するには、**Generate Code For Current Experiment** ボタンをクリックします。

または

- **Extras → Generate Code → For Current Experiment** を選択します。

ハードウェアコンフィギュレーションをインポートする：

- ハードウェアコンフィギュレーションをインポートするには、**Import** ボタンをクリックします。

または

- **Files → Import** を選択します。

ハードウェアコンフィギュレーションをエクスポートする：

- ハードウェアコンフィギュレーションをエクスポートするには、**Export** ボタンをクリックします。

または

- **Files → Export** を選択します。

7.2.2 “Items”リスト

エディタの左側ペーンにツリー構造で表示される“Items”リストは、ハードウェアシステムの構成を表すものです。ここにハードウェアを追加することにより、システム構成を定義します。

ここで、アイテム（"Item”）という語は、ハードウェアシステムを構成する各階層の 1 つの要素を表します。つまり、ボード単体を示す場合もあり、またある特定の機能を実現するための 1 つのユニット全体を表す場合もあります。

注記

ハードウェアの構成は、ハードウェアツリーのすべての末端に“Device”タイプのアイテムが定義されていないと、完全に定義されたことにはなりません。

“Items”リスト内の各アイテムを選択すると、そのアイテムについての情報や設定内容が右側の表に表示されます。

7.2.3 コンフィギュレーション用タブ

アイテムリストのどのアイテムが選択されているかにより、ウィンドウ右側部分に表示されるアイテム設定用のタブの種類が異なります。

“Device”タイプのアイテムが選択されている場合は 4 つのタブが表示され、それ以外の場合は“Globals”というタブだけが表示されます。

各タブの内容は以下のとおりです。

- “Globals”タブ

  グローバルオプション、つまりアイテム全体に共通なオプション項目（例、I/O ボードの VME バスアドレス）が含まれます。

- “Groups”タブ

  シグナルグループに関するオプション項目（例、マルチチャンネル A/D コンバータ用のゲインファクタや CAN メッセージの識別子）が含まれます。

  注記

  1 つのシグナルグループ内のすべてのシグナルの転送方向（send または receive）は同じで、同じタイミングで取得されます。

- “Signals”タブ

  シグナルごとのオプション項目（例、変換式）を設定するために使用されます。

- “Mappings”タブ

  シグナルと ASCET メッセージの対応関係を定義するために使用されます。HWC モジュールと他の部分との間の RTIO データフローもここで設定します。

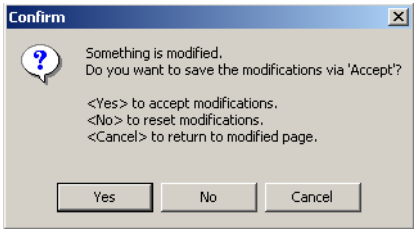
各タブの設定についての詳細は、118 ページの「コンフィギュレーションタブ」を参照してください。

さらに、各タブには **Accept** および **Reset** という 2 つのボタンがあります。これらのボタンは以下のように使用します。

Accept ボタン：

タブが初めて開かれたときや再度開かれたときには、各オプション項目に設定されているデータのコピーが作成され、その内容が表示されます。いずれかのセルを編集すると、そのコピーされたデータが変更されます。

そのため、編集終了後は、**Accept** ボタンをクリックして変更後のデータを元のデータに明示的に書き込む必要があります。書き込みを行わずにタブを閉じようとすると、システムはデータ書き込みを行うようにユーザーに要求します。



“Items”リスト内の複数のアイテムの依存関係を考慮しなければならない場合があるので、上記のメカニズムによって常にデータの整合性を保証することが必要となります。

Reset ボタン：

このボタンで、前回 **Accept** をクリックした後に行われたすべての修正内容が取り消されます。

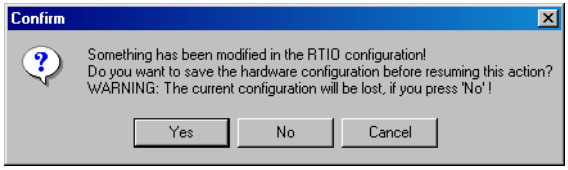
7.2.4 メインメニュー

本項では、各メニューに含まれるメニューコマンドについて説明します。

“File”メニュー

- **File → New**

  新しいハードウェアコンフィギュレーションを生成します。既存のハードウェアコンフィギュレーションを変更した後にその変更内容がまだ保存されていない場合、システムはユーザーに対して以下の質問を行います。

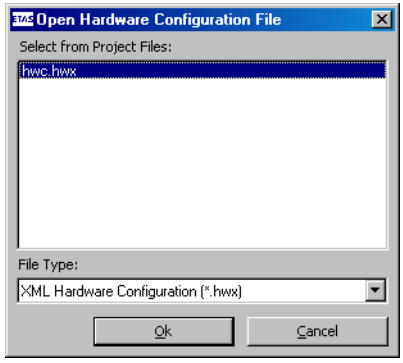


  Yes をクリックして確定すると、プロジェクトファイルコンテナ内のコンフィギュレーションが保存されます。**No** をクリックすると、たった今作成されたコンフィギュレーションは削除され、新しいハードウェアコンフィギュレーションが生成されます。**Cancel** はハードウェアコンフィギュレーション生成処理を打ち切ります。

- **File → Open...**

  “Open Hardware Configuration File”ダイアログボックスが開きます。“File Type”コンボボックスでフォーマット（`*.HWX` または `*.HWC`）を選択すると、プロジェクトファイルコンテナに含まれるそのフォーマットの

ファイルがすべて“Select from Project Files”リストに表示されるので、ファイルを選択して **OK** をクリックすると、そのファイルが HWC エディタにロードされます。



注記

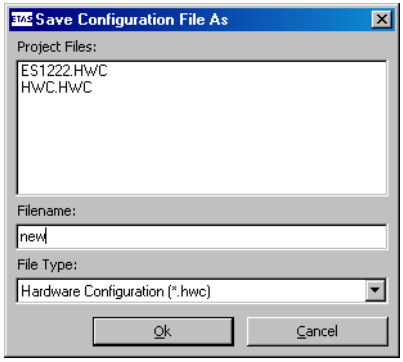
`HWC.HWX`（または `HWC.HWC`）というファイルは特別なファイルです。このファイルは、HWC エディタでコード生成が開始されると必ず暗黙的に生成、または上書きされます。

- **File → Save**

  ハードウェアコンフィギュレーションを同じ名前で ASCET プロジェクトのファイルコンテナに保存します。

- **File → Save As ...**

  ハードウェアコンフィギュレーションを新しい名前で ASCET プロジェクトのファイルコンテナに保存します。“Save Configuration Files as”ダイアログボックスでフォーマット（*.HWX または *.HWC）を選択し、“Filename”フィールドに新しいファイル名を入力します。



  “Project Files”リストには、選択されているタイプの既存のファイルがすべて表示されるので、この中のいずれかのファイルに上書きすることもできます。ファイルを選択するとそのファイル名が“Filename”フィールドに表示され、**OK** をクリックすると、現在のハードウェアコンフィギュレーションがそのファイルに上書きされます。

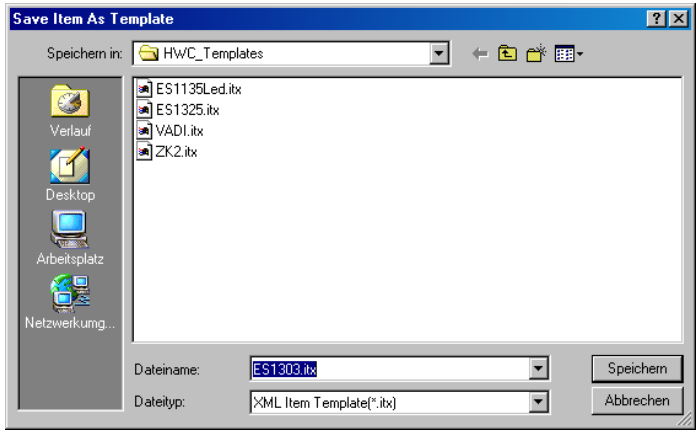
- **File → Save Item as Template ...**

  選択されているアイテムの設定を「アイテムテンプレート」として保存し、他の同じタイプのアイテムに同じ設定を適用できるようにします（108 ページの「Extras → Options ...」メニューの **Use Item Templates** を参照してください）。

  注記

  ASCET-RP V5.2 より、アイテムテンプレートは、デフォルト状態においては XML フォーマット（`*.itx`）で保存されるようになりました。
  必要に応じて既存の `*.itp` フォーマットを選択して使用することもできますが、ASCET-RP の将来のバージョンでは `*.itp` フォーマットはサポートされなくなるため、できる限り XML フォーマットをお使いいただくことをお勧めします。

  このメニューコマンドを選択すると“Save Item As Template”ダイアログボックスが開くので、“File Type”コンボボックスでフォーマット（`*.itx` または `*.itp`）を選択し、“Filename”フィールドにファイル名を入力します。必要に応じて任意のパスを選択することも可能です。



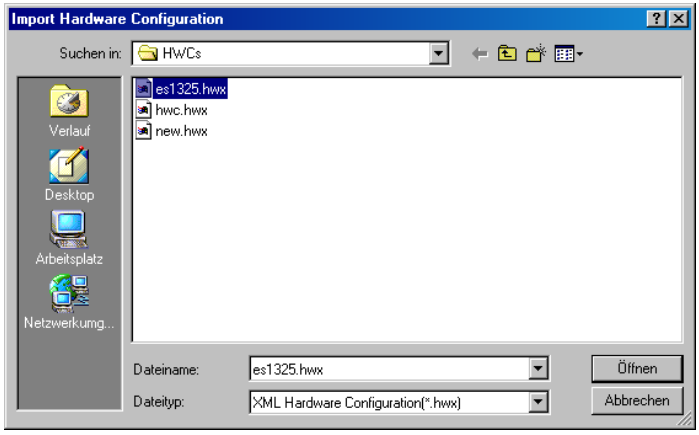
  注記

  ダイアログボックスが開いた時には、“Options”の“Template Path”で設定されたデフォルトパスが表示されますが、このパスは任意に変更できます。
  ただし、表示されるファイル名は絶対に変更しないでください。

- **File → Import ...**

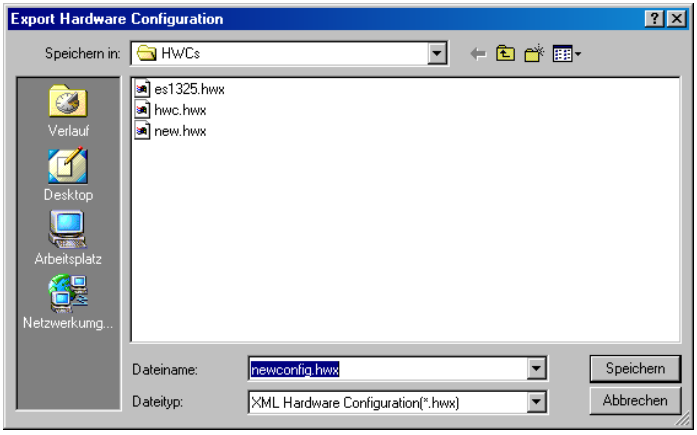
  ハードウェアコンフィギュレーションはファイルから読み込むことが可能です。このためには、“Import Hardware Configuration”ダイアログボックスでファイルタイプ（*.HWX または *.HWC）を選択し、さらにパスを選択してファイルを指定します。

  ただしここで読み込めるハードウェアコンフィギュレーションは、ASCET TIP Exp V4.0.0、またはそれ以降のバージョンでエクスポートされたものに限ります。



- **File → Export ...**

  ハードウェアコンフィギュレーションは、フォーマットと名前を指定してファイルにエクスポートすることができます。ファイルフォーマットは 3 種類のものから選択できます。



  ASCET-RP 環境内でハードウェアコンフィギュレーションの交換を行うには、`*.HWX` または `*.HWC` フォーマットを使用します。もう 1 つの `*.csv` フォーマット（値がカンマで区切られたフォーマット）は、ドキュメント作成やスプレッドシートアプリケーションに利用するためのものです。

  注記

  デフォルト状態において、ハードウェアコンフィギュレーションは XML フォーマット（`*.hwx`）で保存されます。
  必要に応じて既存の `*.hwc` フォーマットを選択して使用することもできますが、ASCET-RP の将来のバージョンでは `*.hwc` フォーマットはサポートされなくなるため、できる限り XML フォーマットをお使いいただくことをお勧めします。

- **File → Exit**

  HWC エディタを閉じます。修正後のハードウェアコンフィギュレーションを保存せずにエディタを閉じようとすると、警告メッセージが表示されます。

- **Edit → Copy → Item(s)** （**<Ctrl> + <C>**）

  選択されたアイテムとそのサブツリー（存在する場合）のコピーを作成し、HWC エディタの内部バッファに転送します。

  このバッファの内容は、HWC エディタが開いている間のみ有効です。

- **Edit → Copy → Item(s) For Export**

  このメニューコマンドの機能は **Copy → Item(s)** と同様ですが、コピーされた内容が外部ファイル（`*.HSX` または `*.HWS`）に書き込まれる点が異なります。この機能は、コンフィギュレーションの一部を交換するような場合に便利です。

  注記

  ASCET-RP V5.2 より、サブツリーは、デフォルト状態においては XML フォーマット（`*.hsx`）で保存されるようになりました。
  必要に応じて既存の `*.hws` フォーマットを選択して使用することもできますが、ASCET-RP の将来のバージョンでは `*.hws` フォーマットはサポートされなくなるため、できる限り XML フォーマットをお使いいただくことをお勧めします。

- **Edit → Copy → Item Data**

  アイテムのデータを内部バッファにコピーします。サブツリー全体のコピーは行われません。この機能を利用して、2 つのアイテム間で設定内容を交換することができます。

- **Edit → Paste → To Selected Item** （**<Ctrl> + <V>**）

  このメニューコマンドは、**Copy → Item(s)** でコピーされたアイテム（またはサブツリー）を、現在選択されているアイテムの下の階層に貼り付けます。この機能を利用するには、以下の条件が満たされている必要があります。

  1. コピー元と貼り付け先のアイテムのタイプが互いに一致していなければなりません。ただし例外として、システムコントローラアイテム間でサブツリーを交換することは可能です。
  2. 貼り付け先のアイテムが、アイテムまたはサブツリーを追加できる状態でなければなりません（**Add Hardware Item** と同様の条件が適用されます）。

- **Edit → Paste → To Selected Item From Import**

  機能は **Paste → To Selected Item** と同じですが、ファイルから読み込まれた内容が貼り付けられます。**Edit → Paste → To Selected Item** と同じ制約条件があります。

- **Edit → Paste → Item Data**

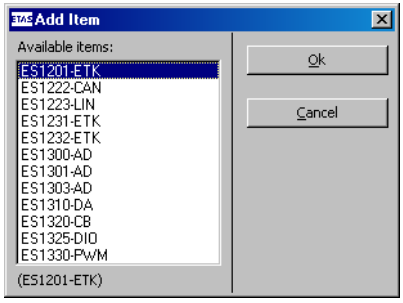
  現在選択されているアイテムに、**Copy → Item Data** でバッファに書き込まれたデータを上書きします。

  前の 2 つの **Paste** メニューコマンドとは異なり、互換性のある異なるタイプのアイテム間でデータを交換することもできます（互換性のあるアイテムのリストは後で紹介します）。この機能を実行するには、データのコピー先となるアイテムのタイプが、コピー元のアイテムと同じ、あるいは互換性のあるものでなければなりません。つまり、両方のアイテムのデータ構造が同じで、“Groups”、“Signals”、“Mappings” の各タブに表示される設定データの行数が一致している必要があります。

- **Edit → Add Item ...** （**<Ins>**）

  新しいハードウェアアイテムをハードウェアコンフィギュレーションに追加します。

  選択リストが表示され、現在の階層で選択可能なアイテムがリストアップされます。



  ここで選択されたアイテムが、ハードウェアコンフィギュレーションに追加されます。

  注記

  コンフィギュレーションによっては、必要なリソース（ポート、スロット）がすでに使用されている場合や特別なシーケンスが必要な場合など、アイテムを正しく選択しても、それが追加されずにエラーメッセージが発行される場合があります。

- **Edit → Delete Item** （**<Del>**）

  選択されたアイテムをハードウェアコンフィギュレーションから削除します。

  階層内の先頭の“Hardware Interface Manager”タイプのアイテムは、削除することはできません。

- **Edit → Move Item Up** （**<Ctrl> + <U>**）

  ハードウェアコンフィギュレーションにおいて、ある階層レベル内のアイテムの並びを変更します。選択されたアイテムが 1 つ上に移動します。

- **Edit → Move Item Down** （**<Ctrl> + <D>**）

  ハードウェアコンフィギュレーションのある階層レベル内のアイテムの並びを変更します。選択されたアイテムが 1 つ下に移動します。

- **Edit → Add Row Before**

  このメニューコマンドは、“Device”タイプのアイテムのうち、シグナルグループ数またはシグナル数を動的に変更できるものについてのみ有効です。

  注記

  システムによってテーブルのサイズが制限される場合があり、その場合、このメニューコマンドは無効になり、グレイアウト表示になります。

  このメニューコマンドを使用できる条件は、以下のとおりです。

  1.“dynamic”機能を持つ“Device”タイプのアイテム（例、CAN-IO デバイス）が選択されていること

  2.“Groups”または“Signals”タブが選択されていること

  3.“No”列（テーブルの第 1 列）の項目が選択されていること

  このメニューコマンドによって、選択されているセルまたは領域の前（= 上）に行が追加されます。

  注記

  この機能を実行するとテーブルの構造が変更されてしまうため、***Edit* → Add *** または ***Edit* → Delete Row(s)** コマンドで実行された変更内容を **Reset** ボタンで取り消すことはできません。

例：
“Signals”タブの選択されているセル領域（No.3 ～ 4）の前に、1 行追加します。これによって、第 3 行（“Signal6”シグナル）が新規に作成されます。

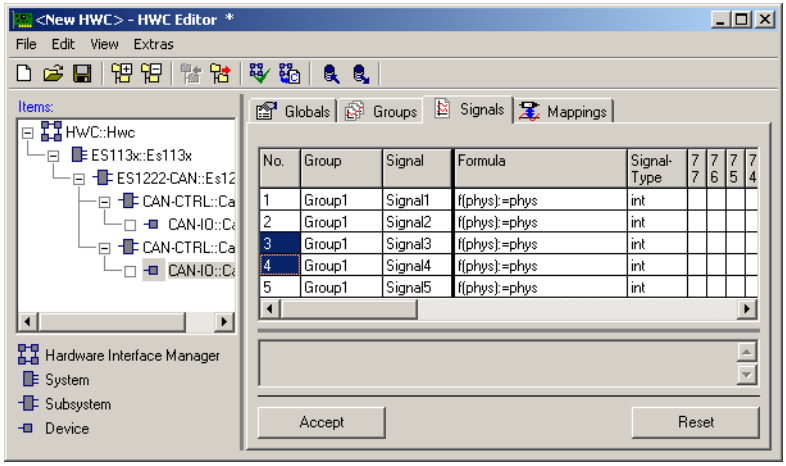


図 7-1　HWC エディタで行の挿入位置の直後の領域を選択したところ

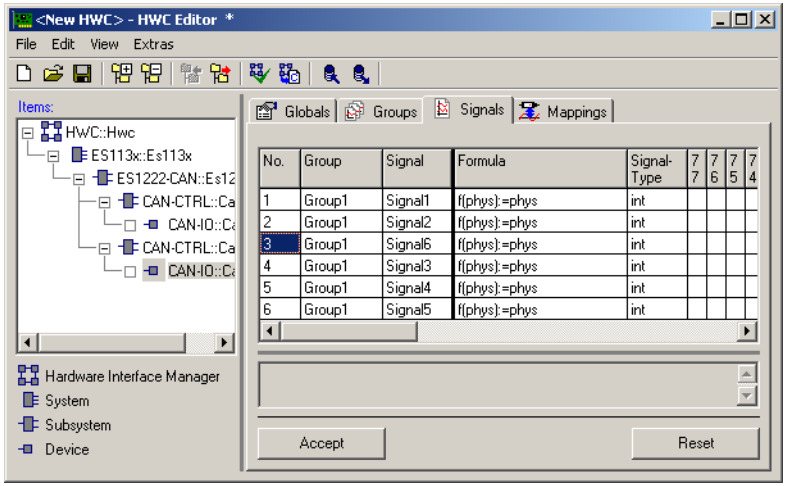


図 7-2　HWC エディタに新しい行（強調表示されている“Signal6”シグナルの行）が挿入されたところ

- **Edit → Add Row After**

  Add Row Before と同様に、選択されたセルまたは領域の**後ろ**に 1 行が追加されます。

- **Edit → Add Multiple Rows Before**

  Add Row Before と同様の機能ですが、ここでは一度に複数の行を追加することができます。挿入する新しい行の数は、ダイアログボックスで指定します。

- **Edit → Add Multiple Rows After**

  Add Multiple Rows Before と同様に、選択されたセルまたは領域の**後ろ**に複数の行が追加されます。

- **Edit → Delete Row(s)**

  “No”列が強調表示されている行（1 行または複数行）を削除します。

  注記

  システムの都合上、テーブルにはシグナルグループまたはシグナルが最低 1 つは残っている必要があるため、最後の行を削除しようとするとエラーメッセージが表示されます。

“View”メニュー

- **View → Expand All**

  “Items”リスト内のツリー構造をすべて展開表示します。

- **View → Collapse All**

  “Items”リスト内のツリー構造をすべて省略表示します。

- **View → Show All**

  表示できるすべてのオプション項目と列を表示します。

  デフォルト状態においては、テーブル表示を見やすくするため、通常は使用されない一部の項目または列は、表示されません。

  注記

  Accept ボタンでテーブルを確定してハードウェアコンフィギュレーションを保存すると、再度ロードされるテーブルは、保存された時とまったく同じ状態で表示されます。つまり、新しく表示された項目／列もそのまま表示されます。

- **View → Hide All Hidable**

  上の **Show All** とは反対に、非表示にできるすべてのオプション項目／列を非表示にします。

- **View → Hide Selected**

  現在選択されているオプション項目または列が非表示にできるものであれば、それを非表示にします。

“Extras”メニュー

- **Extra → Map Item Signals**

  “Automatic Mapping”ダイアログボックス（183 ページの「Automatic Mapping」を参照してください）を開きます。

- **Extra → Check Item**

  選択されているアイテムに関するすべての設定内容を明示的にチェックし、さらに現在の ASCET プロジェクトに対する妥当性もチェックします。このチェックは、RTIO のコードが生成される前にも暗黙的に行なわれます。

  このチェックが正常に終了すると、RTIO コード生成が可能となります。ワーニングは許容されますが、その内容には充分な注意が必要です。

- **Extra → Check Hardware Configuration**

  ハードウェアコンフィギュレーション内の設定内容の整合性に関して可能な限りのチェックを明示的に行いながら、現在の ASCET プロジェクトに照らしたチェックも行います。このチェックは、RTIO コード生成が行われる前にも、必ず暗黙的に行われます。

  RTIO コード生成は、このチェックが正常に完了したときに限り可能です。ワーニングが発行された場合もコードは生成されますが、ワーニングの内容には充分な注意が必要です。

- **Extras → Generate Code → For Current Experiment**

  現在 ASCET プロジェクトエディタ上に開いている実験用の RTIO コード生成を開始します。これは以下の手順で行われます。

  1.HWC モジュール全体の内容が削除されます（その HWC モジュールにより生成されたすべてのプロセスがタスクリストから削除されます）。
  2.必要なすべてのエレメント（メッセージ、パラメータ、変数など）が HWC モジュール内に生成されます。
  3.HWC モジュールのヘッダおよびプロセス用の C コードが生成されます。
  4.HWC モジュールのすべてのプロセスが、適切なタスクに分割されます。

- **Extras → Generate Code → For Phys. and Quant. Experiment**

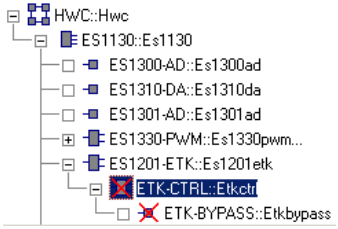
  上の **... Generate Code → For Current Experimen** と同じ方法で、物理実験および量子化実験用のコードが HWC モジュール内に生成されます。

- **Extras → Generate Code → For Phys., Quant. and Impl. Experiment**

  これも **... Generate Code → For Current Experimen** と同じ方法で、物理実験、量子化実験、および実装実験用のコードが生成されます。

- **Extras → Item Code Generation On/Off**

  選択されているアイテム以下のすべてのアイテムについて、コード生成を行なうかどうかを指定します。デフォルト状態においては全アイテムについてコード生成が行なわれるようになっているため、このメニューコマンドを最初に選択した時にはそのアイテムのコード生成は無効になり、アイテムツリー上に赤い X 印が付きます。



  上のように表示されている状態でコード生成を行なうと、X 印の付いたアイテムは無視されます。

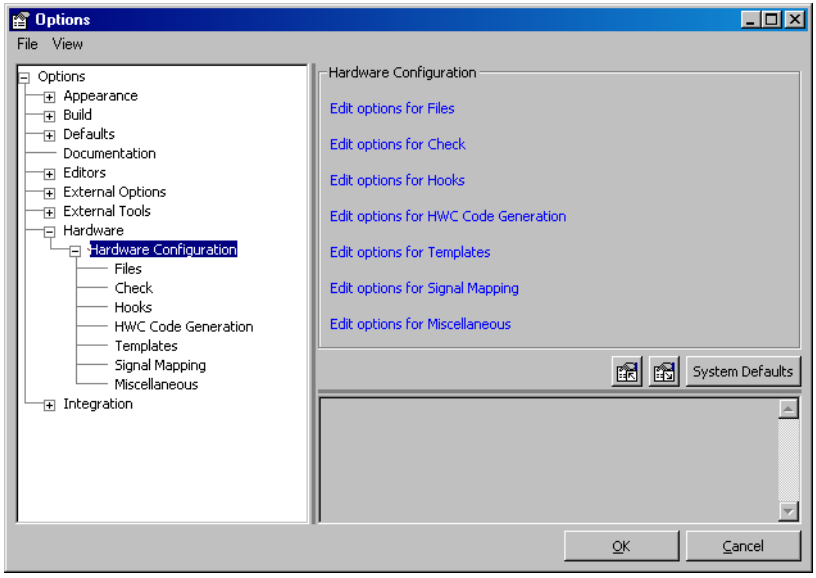
  ここでもう一度このメニューコマンドを選択すると、X 印が消えてコード生成が有効になります。

- **Extras → Clear HWC Module**

  HWC モジュールの全内容（エレメントおよびプロセス）を削除します。

- **Extras → Options ...**

  オプションウィンドウの“Hardware Configuration”ノードを開き、HWC エディタに関する設定を行います。



各サブノードで設定するオプションを、以下に説明します。

“Files”ノード

Default storage format: XML (.hwx and *.itx) / Legacy (*.hwc and *.itp)*

ハードウェアコンフィギュレーションファイルとアイテムテンプレートファイルのデフォルトフォーマットを選択します。

Validate XML files

`*.hwx` または `*.itx` ファイルの読み込み時に XML コードの構文をチェックするかどうかを指定します。このオプションがオンになっていると、ファイルの読み込み時に構文エラーが見つかった場合、読み込み処理が中断されます。

Verify XML Checksum

`*.hwx` または `*.itx` ファイルの読み込み時にチェックサムの照合を行うかどうかを指定します。この照合により、読み込まれたファイルが、ASCET-RP で生成された後に内容が変更されていないかどうかがチェックされます。このオプションがオンになっていると、照合エラー発生時にはワーニングメッセージが出力されます。

“Check” ノード

After loading

新しいハードウェアコンフィギュレーションがロードされた時に、自動的に暗黙的なチェックが行なわれるようにするかどうかを指定します。

Before saving

ハードウェアコンフィギュレーションが保存される前にそのハードウェアコンフィギュレーションについてのチェックが自動的に行なわれるようにするかどうかを指定します。

After "Accept"

Accept ボタンがクリックされたときに、設定内容の整合性チェックが自動的に行なわれるようにするかどうかを指定します。パフォーマンス上の理由から、チェックは現在選択されているノード内のデータについてのみ行われます。

“Hooks” ノード

Post Check Hook

Check Hardware Configuration によるチェックが行われた後に、ユーザー定義されたチェックが実行されるようにするかどうかを指定します。

> **注記**
>
> 以下の 3 つのオプションは、**Post Check Hook** オプションがオンになっている場合にのみ有効です。

Post Check Command

ユーザー定義されたチェックコマンドの、パスとファイル名

Post Check HW Configuration File

コマンドを呼び出す前にハードウェアコンフィギュレーションを保存する場所を指定します。

Post Check Log File

ユーザー定義のチェックによって発行されるワーニングとエラーが保存されたファイルを指定します。ログファイルに保存するメッセージは、以下のフォーマットを使用してください。

```
<log message entry> --> <log message type> ': ↵
     (' <location> ') >>> ' <log message>

<log message type> --> {ERROR|WARNING}

<location> --> <hwc item name> '::'
     <hwc item type> '"' <tab name> [' \ '    ↵
     <column or row name> [' \ ' <row number>]]
```

```
<hwc item name> --> HWC item name as in       ↵
      hardware configuration
<hwc item type> --> HWC item type as in       ↵
      hardware configuration
<tab name> --> {'Globals' | 'Groups' |        ↵
      'Signals' | 'Mappings'}
<row or column name>--> Name of the row (for  ↵
      'Globals' tab) or column (other tabs)   ↵
      to which the message belongs
```

コマンド実行後、指定されたログファイルが存在するかどうかがチェックされ、存在しない場合はエラーが発生します。

Post Generation Hook

ユーザー定義のコード生成を行うかどうかを指定します。

注記

以下の 3 つのオプションは、**Post Generation Hook** オプションがオンになっている場合にのみ有効です。

Post Generation Command

ユーザー定義されたコード生成コマンドの、パスとファイル名

Post Generation HW Configuration File

ユーザー定義されたチェックコマンドの、パスとファイル名

Post Generation Log File

ユーザー定義のチェックによって発行されるワーニングとエラーが保存されたファイルを指定します。ログファイルに保存するメッセージのフォーマットは、“Post Check Log File”のものと同じです。

“HWC Code Generation”ノード

Automatic repair

ハードウェアコンフィギュレーション内で見つかったエラーのうちのいくつかを自動的に修復する機能を有効にするかどうかを指定します。ただし現時点では、この修復機能が実際に働くケースは、ごくまれです。

Store HW Cnfiguration in generated C-Code

このオプションがオンになっていると、XML フォーマットで保存されたハードウェアコンフィギュレーションが、HWC モジュールの C コードに、コメントとして追加されます。

> **注記**
>
> ハードウェアコンフィギュレーションのサイズは、非常に大きくなる場合があります。

Log Level: silent / brief / verbode

HWC コード生成においてログ生成が必要かどうか、また必要な場合はその内容の詳細さを指定します。

“Templates” ノード

Use Item Templates:

アイテムテンプレートを使用するかどうかを指定します。

アイテムテンプレートは、システムにより指定されるデフォルト値に個々の設定を上書きするために使用されます。Template オプションがオンになっていると、**Edit → Add Item** でアイテムを追加するたびに、システムにより指定されたデフォルト値が、**Edit → Save Item As Template** でテンプレートディレクトリに保存された値によって上書きされます。

Template Path

アイテムテンプレートを格納するディレクトリを選択します。

“Signal Mapping” ノード

Message Creation Target: Project / Module

自動マッピング実行時に生成される ASCET メッセージを、プロジェクト内に生成されるようにするか（`Project`：デフォルト）、またはインクルードされるモジュール内に生成されるようにするかを指定します。

Module Instance Name

このオプションは、"message Creation Target” が `Module` に設定されている場合にのみ有効です。

ここにはメッセージを生成するモジュール名を入力します。プロジェクト内のすべてのモジュール（プロジェクトに直接含まれるものと、他のモジュールに含まれるもの）から指定できます。モジュールに含まれるモジュールを指定する場合、2 つのモジュール名をドット "." で区切ります。たとえば、モジュール A に含まれるモジュール B の指定する場合は `A.B` と入力します。

指定されたモジュールがプロジェクトに含まれていない場合、以下のようなエラーメッセージが出力されます。

```
Can't add Message '<msg name>', because module
'<module name>' doesn't exist in project!
```

Create Signal Type specific Message

ASCET メッセージのコード生成時に、シグナルのデータ型が考慮されるようにするかどうかを指定するオプションです。このオプションがオンになっていると、メッセージのモデル型は、シグナルのデータ型またはシグナルに応じて、以下のように決定されます。

デバイス	シグナルのデータ型（CAN/LIN/ETK）またはシグナル	メッセージのモデル型
CAN-IO, CAN-Bypass, LIN-IO, ETK-BYPASS, ETK-BYPASS-ADV	real	cont
	uint[a]	udisc
	int[a], (s)int[b]	sdisc
	Bool, uint （1 つのビットのみが 1 にセットされる）	log
ES1300-AD, ES1303, ES1310, ES1330-PWM	（全シグナル）	cont
ES1135-LED, ES1320-CB/DIO, ES1325-LED	（全シグナル）	log
ES1325-INPUT, ES1325-OUTPUT	Error State, State	log
	Active Time, Additive Active Time, Duty Cycle, Frequency, Inverse Duty Cycle, Inactive Time, Period Duration, Ratio Active Time/Inactive Time, Ratio Inactive Time/Active Time	cont
	Counter Value, No. of Periods	udisc

a. バイパスシグナルの変換式が恒等式でない場合、モデル型 cont の ASCET メッセージが生成されます。

b. CAN-IO および LIN-IO でのみ使用されます。

このオプションがオフになっていると、常に cont 型のメッセージが生成されます。

Create Signal Direction specific messages

ASCET メッセージのコード生成時に、シグナルの方向が考慮されるようにするかどうかを指定するオプションです。このオプションがオンになっていると、メッセージタイプは、シグナルの方向に応じて以下のように決定されます。

転送方向	モジュール／プロジェクト内に生成されるメッセージ
send（送信）	送信メッセージ
receive（受信）	受信メッセージ

このオプションがオフになっていると、常に送受信メッセージが生成されます。

“Miscellaneous”ノード

Data digits after decimal point: 1...6

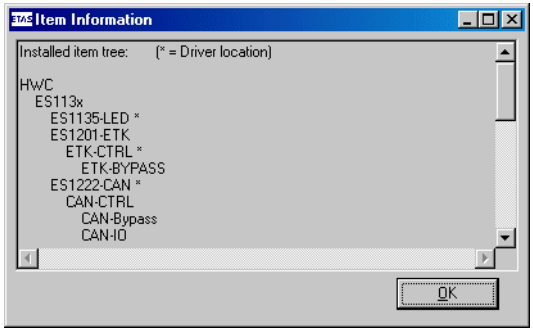
Data digits before decimal point: 1...9

これら 2 つのオプションは、デバイスの“Mappings”ノードの“Data”列の値を表示する際に使用されるフォーマット（桁数）を定義します。

表記：<Data digits before...>.<Data digits after...>

- **Extras → Installed Items ...**

  このメニューコマンドは、その時点でシステムがサポートしているすべてのアイテムおよびその階層的な従属関係をリストアップします。

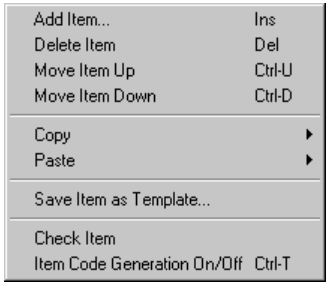


- **Extras → Search for Boards**

  ハードウェアの自動検出処理（114 ページの「ハードウェアの自動検出」の項を参照してください）を開始します。

7.2.5 ショートカットメニュー（“Items" リスト）

“Items“リスト内では以下のショートカットメニューを使用できます。



このショートカットメニューに含まれるメニューコマンドは、HWC エディタのメニューバーのメインメニューにも含まれています。各コマンドの機能は前述のとおりです。

7.3 ハードウェアの自動検出

ASCET-RP V5.5 では、HWC エディタにハードウェアの自動検出機能が追加され、システムに接続されている I/O ボードとその機能の概要が一覧表示されるようになりました。表示されたハードウェアは、任意に選択してハードウェアコンフィギュレーションに追加することができます。

ハードウェアの自動検出処理を開始する：

- “Items”フィールドで、アイテムを選択します。

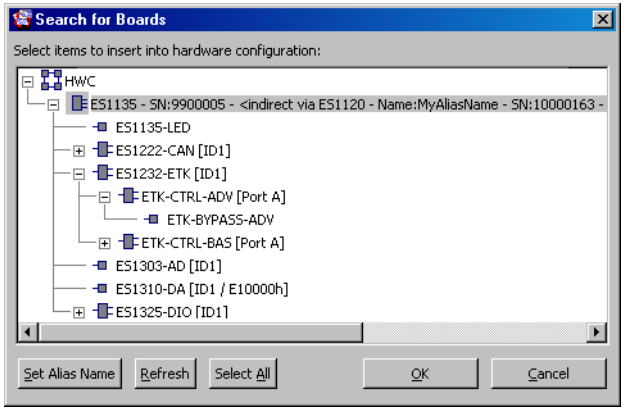
- **Search for Boards** ボタンをクリックします。

または

- **Extras → Search for Boards** を選択します。

  検出されたハードウェアが“Search for Boards”ダイアログボックスに表示されます。

“Search for Boards”ダイアログボックスは、2.2.1 項に説明されているハードウェア選択ダイアログボックスに似ています。



- “Select items to insert into hardware configuration”表示フィールド

  メインアイテム `HWC`（シンボル）の下に、PC に接続されているすべてのシミュレーションコントローラ（ES113x、シンボル）が一覧表示されます。シミュレーションコントローラのラベルには、ES113x の接続方法（`direct` または `indirect`）などの情報が含まれます（20 ページ参照）。

  シミュレーションボードの下には、接続されている各種ボード（シンボル）、およびその下位のサブシステム（シンボル）とデバイス（シンボル）は、表示されます。

- **Set Alias Name** ボタン

  このボタンで、ES113x または ES1120 に任意の名前を割り当てることができます。

- **Refresh** ボタン

  このボタンをクリックすると、表示フィールドの内容が更新されます。新たに接続したり電源をオンにしたハードウェアが表示され、切り離したり電源をオフにしたハードウェアは表示されなくなります。

- **Select All** ボタン

  表示フィールド表示されているすべてのアイテムが選択されます。

- **OK** ボタンと **Cancel** ボタン

  選択したアイテムを HWC エディタの“Items”リストに追加するには **OK** ボタンをクリックします。設定内容をキャンセルしてダイアログボックスを閉じるには **Cancel** ボタンをクリックします。

一覧表示されたハードウェアをコンフィギュレーションに追加する：

- “Search for Boards”ダイアログボックスで、アイテムを選択します（複数選択可）。

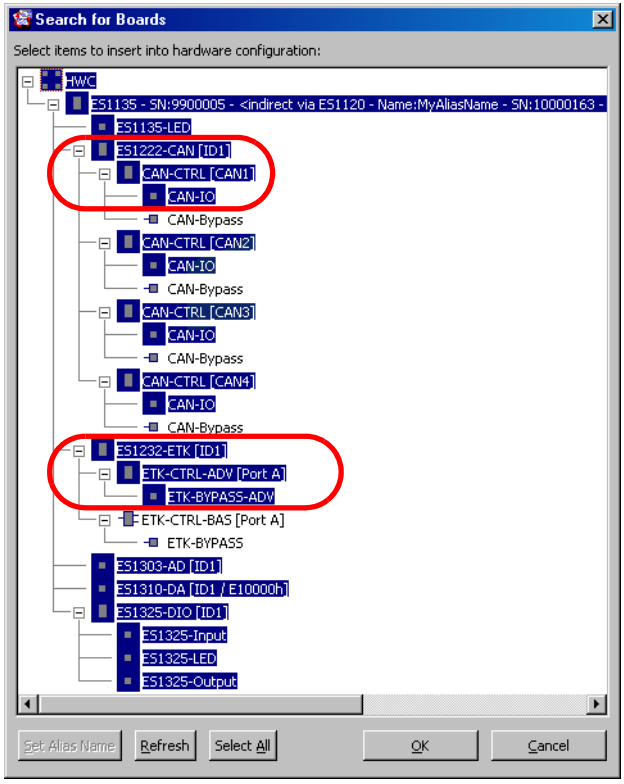
  あるアイテムを選択し、その親アイテムがまだコンフィギュレーションに含まれていない場合、その親アイテムも自動的に追加されます。しかし子アイテムは、すべて個々に追加する必要があります。

  つまり、デバイスなどを追加するとそのデバイスが属するボードも追加されますが、その反対に、ボードを追加しただけではそのボードに属するデバイスは追加されません。

または

- 検出されたハードウェアをすべて選択するには、**Select All** をクリックします。

  複数のサブシステムやデバイスを使用できるボードの場合は（例：ES1222 ボードでは CAN-IO デバイスと CAN-Bypass デバイスが使用可能）、デフォルト構成として定義されているアイテムのみが選択されます。



各ボードのデフォルト構成は、以下のとおりです。

ES1222-CAN	ES1232-ETK
CAN-CTRL	ETK-CTRL-ADV
CAN-IO	ETK-BYPASS-ADV

- **OK** をクリックして、選択されたアイテムをハードウェアコンフィギュレーションに追加します。

7.4 コンフィギュレーションタブ

すべてのアイテムに関するオプション設定は、各コンフィギュレーションタブで行います。“Globals”というタブはすべてのアイテムタイプについて表示され、“Groups”、“Signals”、および“Mappings”タブは、“Device”タイプのアイテムについてのみ表示されます。

7.4.1 全般的な説明

テーブルセルの選択と編集

各タブ上のテーブル内の各セルを選択するには、そのセルをマウスでクリックします。また、矢印キーを使用してテーブル内をナビゲートすることも可能です。

セルが選択された状態でマウスを 1 回クリックするか、または **<F2>** キーを押すと、編集モードになります。入力された内容は **<Enter>** キーで確定され、**<Esc>** キーでキャンセルされます。

また、一部の列（“Formula”列など）は、列内の複数のセルを一度に選択して編集することができます。

連続した複数のセルを一度に編集するには、以下のいずれかの方法で行います。

- 選択範囲の端でマウスの左ボタン押し、そのまま選択範囲の反対側の端までドラッグします。

または

- 選択範囲の端でマウスの左ボタンをクリックし、**<Shift>** キーを押しながら反対側の端でもう一度マウスの左ボタンをクリックします。

  上記のいずれかの操作後、**<Shift>** キーを押したまま最後に選択したセルをクリックすると、編集モードになります。

- カーソルキーで選択範囲の端のセルを選択し、**<Shift>** キーを押したまま上下の矢印キーで複数のセルを選択します。

  <F2> キーを押すと編集モードになります。

連続しない複数のセルを一度に編集するには、以下の方法で行います。

- **<Ctrlt>** キーを押しながら選択したいセルをそれぞれマウスの左ボタンでクリックします。

  <Ctrlt> キーを押したまま最後のセルをダブルクリックするか、または **<F2>** キーを押すと、編集モードになります。

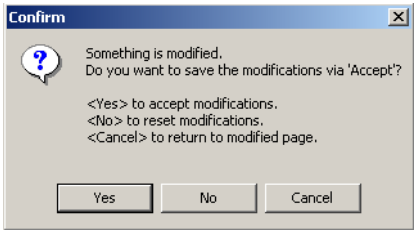
編集を行って **<Enter>** キーを押すと、その内容は選択されたすべてのセルに代入されます。**<Esc>** キーを押すと入力はキャンセルされます

編集

一部のオプション項目は常にロック（グレイアウト表示）されていて編集は行えず、ステータス値を表示するためだけに使用されます。それ以外の項目は、選択されているアイテムやその階層内の他のアイテムのオプション設定に応じて、ロックされる場合もあれば、入力を受け付ける場合もあります。

編集後のページ切り替え

HWC エディタはテーブルの値が変更されたかどうかを認識しているため、**Accept** ボタンで変更内容を有効にする前に別のアイテムまたはタブを選択しようとすると、以下のメッセージが表示されます。



この場合は、**OK** で変更を保存するか、または **Cancel** で選択操作を取り消します。

テーブルのスクロール

テーブルが左右に大きすぎてタブ内に表示しきれない場合、以下のようなスクロールバーが表示されます。

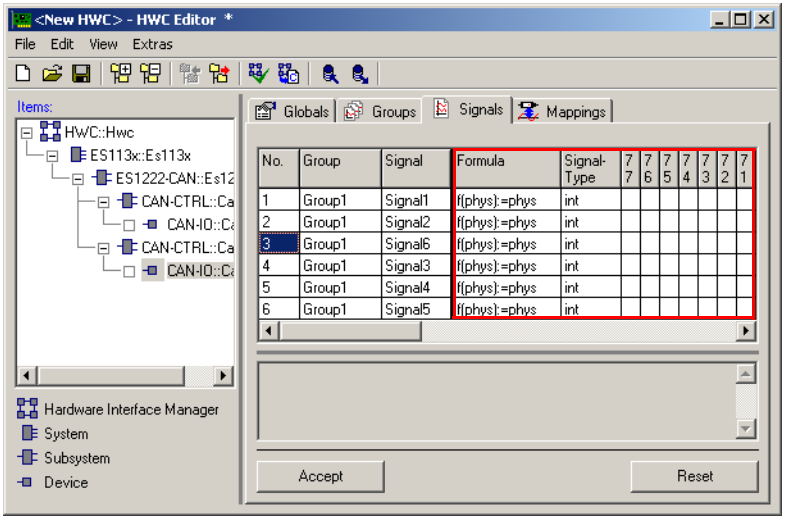


図 7-3　コンフィギュレーションタブ内のスクロール可能領域

スクロールバーを左右に動かすと、テーブル全体ではなく所定の部分（図 7-3 の四角で囲んだ部分）だけが動きます。

注記

テーブルの左側のスクロールされない部分に表示される列が多すぎる場合、右側のスクロール可能な部分がタブ内に入りきらず、スクロールバーを動かしても実際には画面がスクロールされない場合があります。この問題を解決するには、HWC エディタの表示ウィンドウを広げるか、スクロールされない部分に必要最小限の列だけを表示するようにしてください。

列幅の変更

列の縁でマウスの左ボタンをクリックして押し下げたままで左右に動かして、列幅を調整することができます。**Accept** ボタンで確定すると、新たに指定された列幅がハードウェアコンフィギュレーションとともに保存されます。

ハードウェアコンフィギュレーションを再びロードすると、保存された時のレイアウトで表示されます。

ヘルプテキスト

タブの一番下にあるテキストフィールドに、現在選択されているオプション項目についてのヘルプテキストが表示されます。

7.4.2 “Globals”タブ内の共通オプション

本項では、“Globals”タブに表示される共通オプションについて説明します。その他の各アイテム固有のオプションについては、第 10 章「HWC アイテム」の各項で説明します。

Name

アイテムの名前を任意に指定することができます。この名前は、ANSI-C 互換の名前で、ハードウェアツリー内でユニークなものでなければなりません。

このオプションは、すべてのアイテムの“Globals”タブに含まれます。

Init Task

ハードウェアドライバの初期設定を行うタスクを選択します。通常は Init タスクを指定します。Software タスクを指定することも可能ですが、これは例外的なケースであるため、ワーニングが表示されます（67 ページの「準備」という章を参照してください）。

このオプションは、アイテムが低水準ドライバにリンクしている場合にのみ表示されます。

注記

各タスク処理が実行される際に、この初期化タスクを実行してからでないと同じドライバにリンクしている別のタスク（開始タスク、コンフィギュレーションタスクなど）が実行されないようにする必要があり、また“Init”コールは必ず“Exit”コールの後で行なわれなければなりません。それらの条件が守られないと、実験中にランタイムエラーが発生し、ドライバがロックされてしまいます（ドライバロック）。

以下のようなタスクシーケンスが守られるようにしてください。

1. Init Task（初期化タスク）
2. Start Task（開始タスク）
3. Stop Task（停止タスク）
4. Exit Task（終了タスク）

Start Task

ドライバを起動するための独立したタスクを指定することができますが、その必要がない場合には、通常は“Init Task”と同じタスクを指定します。

この“Start Task”は、一部のハードウェアドライバでしかサポートされていません。

注記

タスクの順序を必ず守ってください（120 ページの「Init Task」を参照してください）。

Stop Task

ドライバを停止するための独立したタスクを指定することができますが、その必要がない場合には、通常は“Exit Task”と同じタスクを指定します。

この“Stop Task”は、一部のハードウェアドライバでしかサポートされていません。

注記

タスクの順序を必ず守ってください（120 ページの「Init Task」を参照してください）。

Exit Task

ハードウェアドライバの初期設定を解除するタスクを選択します。これは通常は実際の Exit タスク（非アクティブな Init トリガタスク）を指定します。Software タスクでもかまいませんが、これは例外的なケースなので、ワーニングが表示されます（67 ページの「準備」という章を参照してください）。

この“Exit Task”は、アイテムが低水準ドライバにリンクしている場合にのみ表示されます。

注記

タスクの順序を必ず守ってください（120 ページの「Init Task」を参照してください）。

Config Task

実行時に設定値の変更を行う“Config Task”の選択は、他の ETAS 製品（LabCar）で RTIO を使用する際にのみ行います。ASCET-RP として使用する際は意味を持ちません。

IRQ Handler Task

ここでは必ず Software タスクを選択してください。このタスクは、ASCET プロジェクトエディタ上のタスクリスト内に“Software”タスクとして定義し、“Max. number of Activations”フィールドの値は 2 以上（最大は 50）に設定してください。

割り込みモードで稼働するドライバの場合、この“IRQ Handler Task”が必要です。

注記

このタスクは、アイテムとそれに対応するハードウェアドライバごとに**個別**のものを指定する必要があり、ここで指定されたタスクを他のアイテムやユーザープロセスで共有することはできません。また同じタイプのアイテム（たとえば 2 枚の ES1222）であっても、同じタスクを共有することはできません。

Device Manager Task

このタスクは、実際に交換されるデータの内容（例、エラー処理、バスモニタリング）とは無関係に、確実にデータをドライバに送るためのもので、一部のドライバにのみ必要です。

通常は、ここには Alram タスクを指定します。サイクルタイムとして設定できる値の範囲はデバイスによって異なるため、詳細な情報は、タブ上に表示されるヘルプテキストや本書内の各ハードウェアアイテムについての記述事項を参照してください。

Version

デバイスのバージョンをチェックして低水準ハードウェアドライバと比較を行うためのオプションです。

“Version”はデフォルトでは非表示になっています。

Format

アイテムのフォーマットを表示し、低水準ハードウェアドライバとの互換性チェックを行うためのオプションです。

“Format”はデフォルトでは非表示になっています。

7.4.3 “Groups”タブ内の共通オプション

本項では、“Groups”タブに表示される共通オプションについて説明します。各アイテム固有のオプションについては、第 10 章「HWC アイテム」の各項で説明します。

No

エラーメッセージなどの行番号として使用されます。

Device

アイテム名が表示されます。

“Device”はデフォルトでは非表示になっています。

Group

シグナルグループの名前が表示されます。通常この名前は編集可能ですが、必ず ANSI-C に準拠した名前を使用してください。

注記

1 つのデバイス内の各シグナルグループは、それぞれユニークな名前でなければなりません。

Direction

シグナルグループの転送方向が表示されます。この設定がロックされている場合、つまり表示されたデータがグレイアウトされていて編集できない場合は、転送方向はハードウェアにより規定されるため、ユーザーが変更することはできません（例、A/D ボードは常に“receive”で、D/A ボードは常に“send”になります）。一部のデバイス（例、CAN-IO）についてのみ、ユーザーが転送方向を選択することができます。

Task

データ転送を行うタスクを指定します。また、複数のタスクを指定して、データ転送をその各タスクで行うようにすることもできます。一般的には Alarm タスクまたは Software タスクが使用可能です。一部のデバイスはここでタスクを指定することはできず、データ転送は割り込み制御などの別の方法で行われます。

注記

同じシグナルグループに 2 つ以上のタスクが割り当てられると、付随する RTIO プロセスが複数のタスクに割り当てられることになり、実行時にエラーが発生します。
HWC エディタによって定義される RTIO プロセスはリエントラントな構造になっていないため、複数の RTIO プロセスが同時に実行されることのないように OS コンフィギュレーションを設定する必要があります。
1 つのシグナルグループに複数のタスクを割り当てる場合、この条件を実現する最も簡単な方法は、それらのタスクをすべて協調タスクにする、という方法です。ただしどうしてもプリエンプティブタスクが必要な場合は、ユーザーの責任において適切な OS コンフィギュレーションを設定してエラーを回避してください。

7.4.4 “Signals”タブ内の共通オプション

本項では、“Signals”タブに表示される共通オプションについて説明します。各アイテム固有のオプションについては、第 10 章「HWC アイテム」の各項で説明します。

No

エラーメッセージなどの行番号として使用されます。

Device

アイテム名が表示されます。

“Device”はデフォルトでは非表示になっています。

Group

シグナルグループの名前が表示されます。“CAN-IO”など一部のデバイスについては、シグナルグループへのシグナルの割り当てを自由に定義することができます。

“Group”はデフォルトでは非表示になっています。

Direction

シグナルグループの各シグナルの転送方向が表示されます。

“Direction”はデフォルトでは非表示になっています。

Task

シグナルグループの転送を行うタスクが表示されます。

“Task”はデフォルトでは非表示になっています。

Signal

シグナル名を指定できます。一部のデバイスについては、シグナルの仕様が固定されているため、ユーザーがこれを変更することはできません。シグナル名を入力する場合は、ANSI-C に準拠した名前を使用してください。

注記

1 つのシグナルグループに割り当てられるシグナルの名前は互いにユニークでなければなりません。

Formula

必要に応じてシグナル値に線形変換を割り当てることができます。デフォルトでは 1：1 変換（`f(phys) := phys` という変換式）が選択されています。一部のデバイスについては、式が固定されているため、この列の入力はロックされています。たとえば、ETK-BYPASS デバイスなどがその例で、シグナルは ASAM-MCD-2MC ディスクリプションから生成され、変換式も決められたものが使用されます。

変換式は、シグナル調整処理において使用されます。この処理には、変換、量子化、およびマッピングの 3 つのレイヤが含まれ、RTIO ドライバから受け取ったシグナルグループが、このシグナル調整処理に直接送られます。

変換レイヤでは、シグナルグループが個々の I/O シグナルに分割され、各シグナルは、“Signals" タブのこのオプションで設定された変換式により、ASCET メッセージに対応する物理値に変換されます。

量子化実験または実装実験においては、量子化レイヤによって、これらの物理シグナルが ASCET メッセージに対応して量子化されます。

そして最後のマッピングレイヤで、量子化されたシグナルが ASCET メッセージにコピーされます。

7.4.5 “Mappings”タブ内の共通オプション

本項では、“Mappings”タブに表示される共通オプションについて説明します。各アイテム固有のオプションについては、第 10 章「HWC アイテム」の各項で説明します。

No

エラーメッセージなどの行番号として使用されます。

Device

アイテム名が表示されます。

“Device”はデフォルトでは非表示になっています。

Group

シグナルグループの名前が表示されます。

“Group”はデフォルトでは非表示になっています。

Direction

シグナルグループの各シグナルの転送方向が表示されます。

“Direction”はデフォルトでは非表示になっています。

Task

シグナルグループの転送を行うタスクが表示されます。

“Task”はデフォルトでは非表示になっています。

Signal

シグナル名が表示されます。

ASCET Message

必要に応じて、ここでシグナルと ASCET メッセージの割り当て（マッピング）を行います。ダイアログボックスから、現在のシグナルに割り当てることのできるプロジェクト内のすべての“Exported”メッセージを選択することができます。ただし受信シグナルの場合はエクスポートされる受信メッセージのみが選択でき、また送信シグナルの場合はエクスポートされる送信メッセージのみが選択できます。

“Message selection”ダイアログボックスでは、文字列をキー入力して、目的のメッセージを素早く探すことができます。入力した検索文字列は、ダイアログボックスの一番下に表示されます。

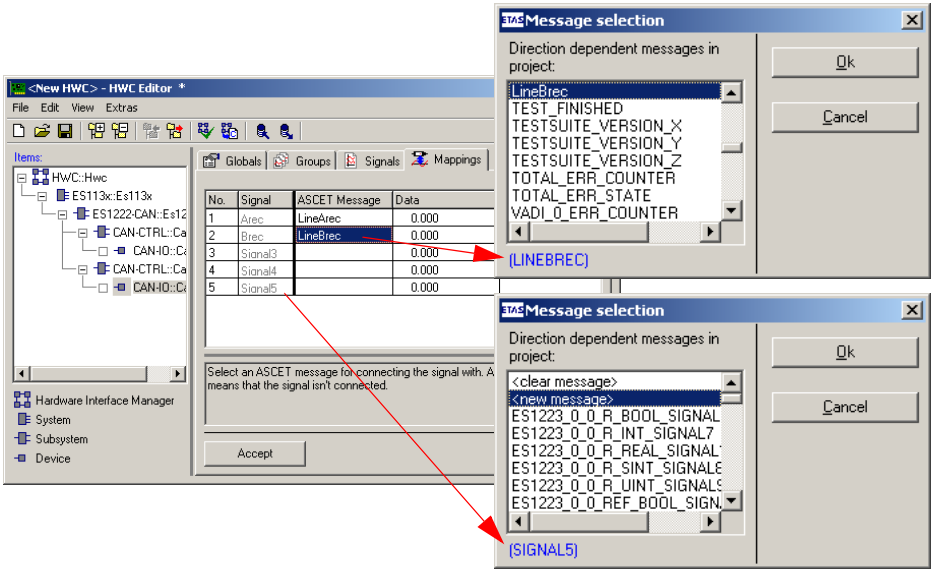


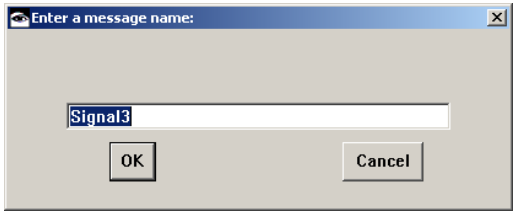
図 7-4 “Message selection”ダイアログボックス

選択ウィンドウを開くと、以下の処理が行われます（図 7-4 を参照してください）。

- “ASCET Message”列にメッセージがある場合には、そのメッセージが検索キーとして使用されます。
- “ASCET Message”列にメッセージがない場合には、シグナル名が検索キーとして使用されます。

“Message selection”ダイアログボックスで <clear message> を選択すると、“ASCET Message”列に入力されているメッセージを削除することができます。

新しいメッセージを生成するには、<new message> を選択します。すると以下のダイアログボックスが開きます。



このダイアログボックスに新しいメッセージ名を入力すると、その名前のメッセージがグローバルな送受信メッセージとしてプロジェクトに登録されます。

Data

送信メッセージのデフォルト値を入力します。RTIO コードジェネレータは、ASCET パラメータにこの値を書き込みます。

Explanation

シグナルについての情報が表示されます。このカラムは一部のアイテムでのみ表示ます。

8 コード生成

RTIO のコード生成は、まず始めにハードウェアコンフィギュレーションの暗黙的なチェックが行われ、それがエラーなしで完了した場合にのみ実行されます（ワーニングのみが発生した場合もコード生成は行われますが、ワーニングの内容に充分注意してください）。

コード生成の処理手順としては、最初に HWC モジュールの全内容（エレメント、コード、およびプロセス）が削除され、その後、新たなコードが生成されます。

注記

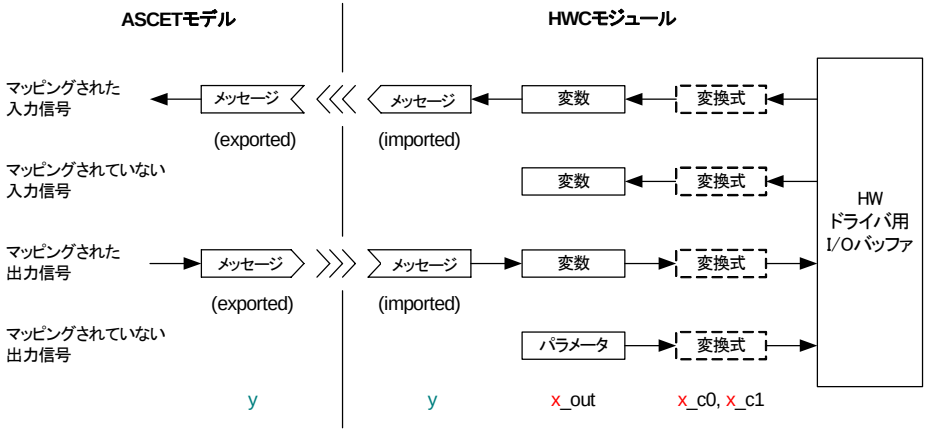
生成されたコードをマニュアル操作で修正しないでください。マニュアル操作で修正してしまうと正しい動作が保証されなくなります。最悪の場合、ハードウェアが損傷してしまう可能性もあります。

8.1 HWC モジュール

本項では、生成されるコードについての概要を簡単に紹介します。ハードウェアやリアルタイム動作についての知識のある方であれば、RTIO リンクの動作原理について要点を押さえることができ、エラーを見つけ出す際にも役立ちます。

8.1.1 エレメント

下図は、プロジェクトと HWC モジュールの間のデータフローをおおまかに示したものです。



メッセージ

HWC エディタの“ASCET Message”列（“Mappings”タブ内）の各項目について、同じ名前のメッセージエレメントが“インポートメッセージ”（前図には“y”と示されています）として作成されます。

I/O インターフェース

ハードウェアドライバと通信を行うすべてのシグナル（`x_out` と表示されます）は、HWC モジュール内で I/O インターフェースを通じてルーティングされます。そのシグナルが送信シグナルと受信シグナルのどちらなのか、あるいはそのシグナルがメッセージにマッピングされているかどうかに応じて、`x_out` という名前の変数またはパラメータが生成されます。

式

シグナルについて線形変換式が指定されている場合には、係数 `c0` および `c1` 用のパラメータが生成されます。

コンフィギュレーションパラメータ

コンフィギュレーション用の値については、それぞれ適合パラメータが生成されます。パラメータの名前は“Item”（および“Group”）プレフィックスとパラメータ識別子で構成されます。パラメータ識別子は各アイテムごとに固有です。

外部コード

割り込みベクタテーブル用のコードおよび割り込みベクタ用のコードは、HWC モジュールの外部コードとして生成されます。

ヘッダコード

低水準ドライバとデータ交換を行うためのデータバッファおよびドライバを初期化するためのデータ構造体は、ヘッダコード内に定義されます。

プロセス

RTIO フレームワークにより生成されるプロセスは、ドライバ制御プロセスとデータ交換プロセスに分類されます。

ドライバ制御用には、以下のプロセスがあります。

- `<item_name>_InitCode_<task_name>_HWCF`
- `<item_name>_StartCode_<task_name>_HWCF`
- `<item_name>_ConfigCode_<task_name>_HWCF`
- `<item_name>_DeviceManagerCode_<task_name>_HWCF`
- `<item_name>_AcknowledgeCode_<task_name>_HWCF`
- `<item_name>_AnalyzeCode_<task_name>_HWCF`
- `<item_name>_StopCode_<task_name>_HWCL`

- `<item_name>_ExitCode_<task_name>_HWCL`

ハードウェアドライバとのデータ交換用には、各シグナルグループごとに以下のようなプロセスが生成されます。

- `<item_name>,<signal_group_name>_<task_name>_HWCF`（受信プロセス）
- `<item_name>,<signal_group_name>_<task_name>_HWCL`（送信プロセス）

HWCF は、<task_name> で指定されるタスク内の既存プロセスの**前**にプロセスが挿入されることを意味します。HWCL は、既存プロセスの**後**にプロセスが挿入されることを意味します。

8.2 プロセスの実行順序

RTIO フレームワークでは、ハードウェア通信の一貫性を保証するため、プロセスは自動的に適切な順でタスクリストに割り当てられます。

注記

この割り当ては、定義されたスキームに従って RTIO コードが生成されるたびに実行されます。この“HWC”プロセスの割り当ては絶対に変更しないでください。変更すると予期しない結果になり、ハードウェアが損傷する可能性もあります。

RTIO フレームワークにより生成されるすべてのプロセスが 1 つのタスクに割り当てられた場合、プロセスシーケンスは以下のようになります。

- Init
- Analyze
- Acknowledge
- DeviceManager
- Start
- Config
- 最初の受信シグナルグループ
- …
- 最後の受信シグナルグループ
- ユーザープロセス
- 最初の送信シグナルグループ
- …
- 最後の送信シグナルグループ
- *Stop
- *Exit

これらのプロセスが、各ハードウェアアイテムについて、階層構造の定義に従って上から順に実行されます。つまり、システムコントローラである ES113x が最初に初期化されてから、下位のハードウェアドライバの初期化が行われます。

ただし上記の“*”が付いているプロセス（“Stop”および“Exit”）は、上記の逆の順序で実行されます。つまり、まず下位のハードウェアドライバの終了処理が行われてから、最後にシステムコントローラの終了処理が行われます。

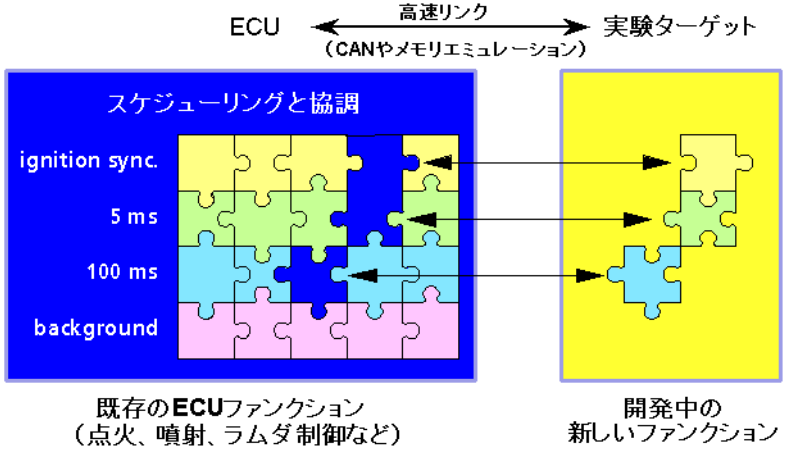
9 ETK バイパス（ES1200/ES1201/ES1231/ES1232）

本章では、ETK バイパスの機能とそれに関連した RTIO パッケージの使用方法について説明します。ここでの記述は ASCET に関する基本的な知識のあるユーザーを対象としているため、ASCET の詳しい使用方法などについては触れられていません。

9.1 ETK バイパスとは？

ETK バイパスとは、ECU の全ファンクションのうちの一部のみをシミュレーションボード、つまり ETAS の実験ハードウェアシステム内に組み込まれた PowerPC シミュレーションプロセッサによって実行させることです。ECU のデータが実験システムに転送され、このデータを使用して、実験ターゲット上にバイパスされたファンクションが実行されます。この部分のデータは容易に変更したり検証したりすることができます。そしてその処理の結果として出力されたデータが ECU に戻され、その後の処理が ECU によって続行されます。

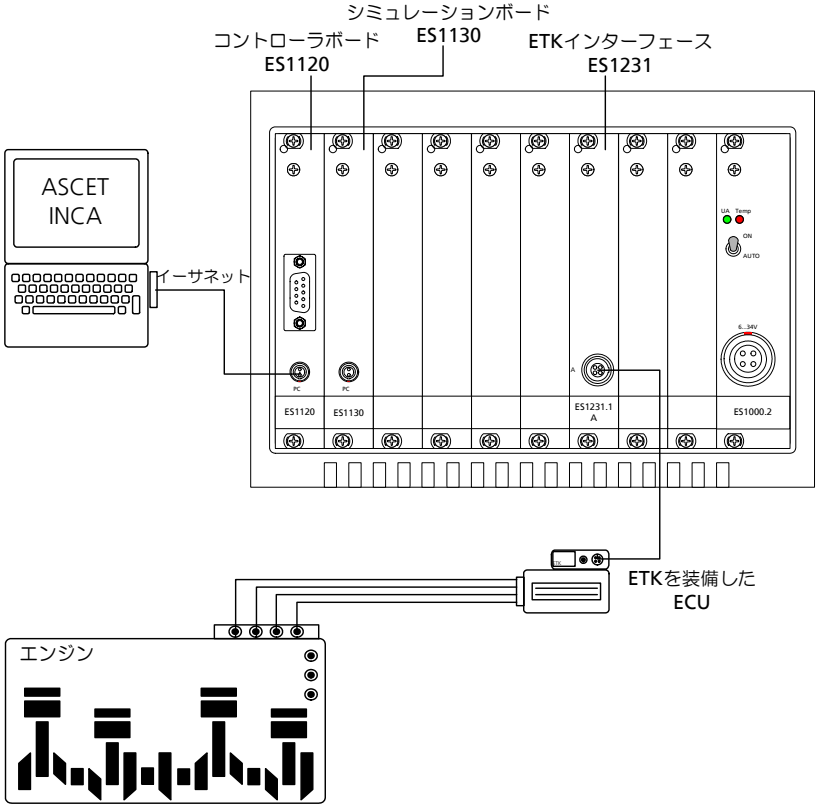
ECU ソフトウェア内でバイパスされる部分に「バイパスフック」と呼ばれるソフトウェアが挿入され、その部分が実験ターゲットにより実行されます。



ETK バイパスは、この ECU と実験ターゲットとの接続に ETK（エミュレータテストプローブ）を使用するもので、ECU と ETK 間はデュアルポート RAM 経由でデータ交換が行なわれます。

9.2 ETK バイパスのハードウェア構成

ES1000.2 システムを使用して ETK バイパス環境を構築するたのハードウェア構成例を下図に示します。



ETK バイパス実験においてバイパス実行されるファンクションは、ASCET で記述します。それを元に、実験システム（ES1000）内の PPC モジュール（PowerPC シミュレーションプロセッサを搭載したシミュレーションボード）上で実行可能なコードが生成され、このコードがホスト PC から実験システムにダウンロードされます。実験システムは、ETK インターフェースボード（ES1200、ES1201、または ES1231）経由で、ECU に取り付けられた ETK に接続します。

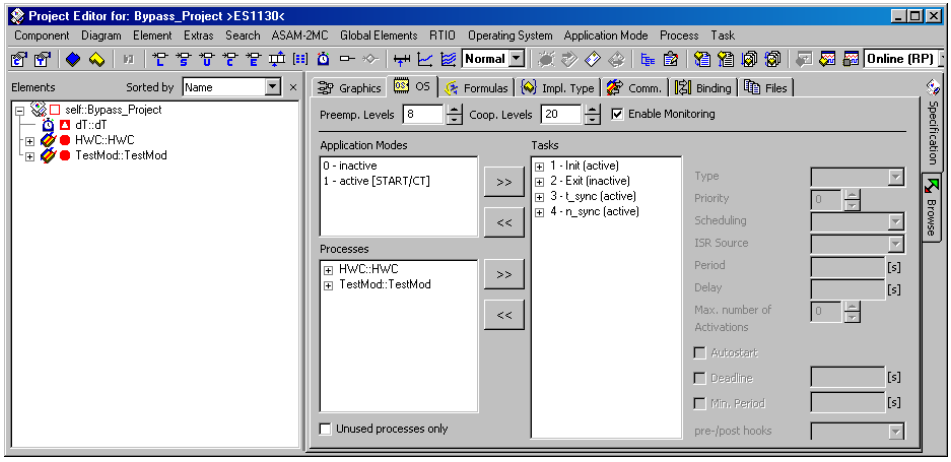
必要に応じて、シミュレーションプロセッサと ECU との間のソフトウェアインターフェースの設定に必要なインターフェースパラメータを、ECU プログラム内に定義することができます。また、新しいソフトウェアバージョンを ECU にダウンロードしたり、バイパスを稼働させながら ECU に関する測定・適合作業を実行することも可能です。

9.3 ETK バイパス用 ASCET プロジェクト

ETK バイパスを使用する ASCET RTIO プロジェクト（つまり RTIO が組み込まれた ASCET プロジェクト）には、以下の点についての注意が必要です。

- ASCET RTIO プロジェクトに、必ず **HWC** という名前の HWC モジュール（5.2.1 項を参照してください）が含まれていること。
- OS エディタのタスクリストに、必ず `Init`（タイプ：Init、アプリケーションモード：active）および `Exit`（タイプ：Init、アプリケーションモード：inactive）というタスクが含まれていること。RTIO パッケージが使用するこれらのタスクの標準的な名前は“Init”および“Exit”です。他のタスク名を使用する場合は、HWC エディタで各コンポーネントを定義する際にタスクをマニュアル操作で割り当てる必要があります。
- OS エディタのタスクリストに、2 つの特殊なタスク（送信タスクおよび受信タスク）が定義されていること。これらのタスクは、シグナルグループに割り当てられたバイパスデータをドライバへ送信、またはドライバから受信します。いずれのタスクも、タスクのタイプは Software です。このタスクの割り当ては、HWC エディタ上でマニュアル操作によって行います。
- RTIO 通信で使用されるすべてのメッセージ（受信メッセージを含みます）が、“Exported”として宣言されている必要があります。
- RTIO 通信用のグローバルメッセージが定義されていること。

これらの条件がすべて設定されると、“OS”タブは以下のようになります。



タスク設定の詳細は、以下のとおりです。

タスク名	アプリケーションモード	トリガモード	優先度	グループ	Max. No. of Act.	周期
Init	active	Init	-	-	-	-
Exit	inactive	Init	-	-	-	-
t_sync	active	Software	16	preemptive	1	-
n_sync	active	Software	17	preemptive	1	-

各属性についての詳細は、『ASCET ユーザーズガイド』を参照してください。

9.4 ETK バイパスのしくみ

バイパスプロジェクトを構築するためには、ECU 側に「バイパスフック」（バイパスファンクションを実行するために修正されたソフトウェア）が組み込まれている必要がありです。このバイパスフックによって、ECU 上で実行されるファンクションとシミュレーションボード上で実行されるファンクションの切り替えが行われます。バイパスフック内には、バイパス入力データをシミュレーションボードに転送し、その出力結果を ECU に戻すために必要な情報が含まれています。

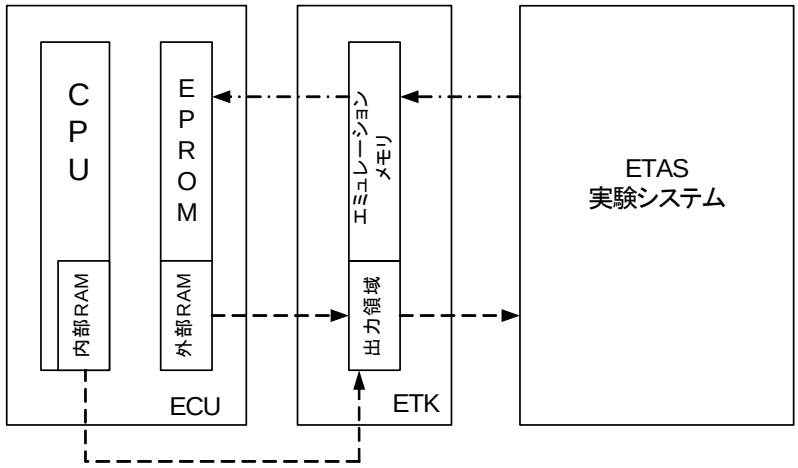
通常、バイパスがイネーブルになっている間、ECU 内のバイパスファンクションも実行されますが、この処理結果は実際には使用されず、シミュレーションボードにおける処理結果、つまり「バイパス出力データ」が有効となります。

「バイパス入力データ」とは ES1000 が ECU から読みとるデータのことで、このデータがバイパスされたファンクションによって処理されます。そしてその処理結果が、バイパス出力データとして ECU プログラムにフィードバックされます。

ETK のメモリは、エミュレーションメモリ領域と出力領域とに分かれています。ECU プログラムはエミュレーション領域にアロケートされ、この領域が ECU の ROM 領域に対応します。ECU は出力領域の内容を直接読みとることができないので、バイパス出力データはエミュレーション領域に格納されます。

出力領域は、バイパス入力データを ECU からシミュレーションボードに転送するためにのみ使用されます。この領域には、ECU の外部 RAM の内容がコピーされ、ETK や ECU の種類によっては、ECU の内部 RAM の内容もコピーされる場合があります。

次の図は、ETK バイパスシステムのデータフローを示したものです。



データはバイパスプロジェクト内をサイクリックに流れます。つまり、ECU からのバイパス入力データが ES1000 システム内で処理され、処理結果であるバイパス出力データが ECU に書き戻されて、それ以降の処理が行われます。バイパス入力データの読み取りは ECU の同期グリッド（時間同期またはは角度同期）によって行われますが、ECU へのデータの書き戻しは非同期に行われます。

9.5 ECU と実験システム間のデータ交換

ETAS 実験システムと ECU の間のデータ転送には、DISTAB データ交換メソッドが使用されます。

ここ数年は、ECU と ETK とのデータ交換には主に DISTAB 12 メソッドが使用されてきました。DISTAB 12 は最大 2 バイト長の測定データをサポートします。

しかし、4 バイト整数や、4 または 8 バイトの実数、または浮動小数点の変数が使用されている場合は、DISTAB 13 を使用する必要があります。DISTAB 13 では、ECU からの 1、2、4、8 バイト長の測定データの取り込みを、フォーマット（符号あり／符号なし、整数／浮動小数点）を問わず行うことができます。

ETK バイパスプロジェクトで DISTAB を扱うためには、さまざまなパラメータを、ECU の現在のソフトウェアバージョンに対応した ASAM-MCD-2MC ディスクリプションファイルに格納して ASCET に渡す必要があります。DISTAB は、ASAM-MCD-2MC ファイル内の **TP_BLOB** の **DISTAB_CFG** セクションに、以下のように定義します。

```
/begin DISTAB_CFG
0xC             /* type of display table:          */
                /* 0xC =DISTAB12, 0xD =DISTAB13    */
0x1             /* Data type of display table:     */
                /* 1=byte 2=word (ECU Data Mode)   */
                /* additional code table for       */
                /* distab13 depending on bus       */
                /* width/bus access (see distab13  */
                /* spec. for more information)     */
MSB_LAST        /* Byte Order: MSB_FIRST/MSB_LAST  */
0x383000        /* Trigger Segment Address         */
0x0             /* Trigger Configuration           */
TRG_MOD 0xB7    /* Dyn. length for TRG_MOD         */
                /* (special code)                  */
/end DISTAB_CFG
```

注記

通常、ASAM-MCD-2MC ファイルにはコメントは含まれません。上記のリストは、注釈のためにコメントを付加したものです。

ASAM-MCD-2MC は、ECU プロジェクト（適合パラメータ、測定変数、変換メソッド、アドレスなど）を記述するための自動車業界標準の規格です。ASAM-MCD-2MC ファイルは、ソフトウェア開発プロセスの出力として、個々のプログラムバージョンごとに作成されます。必要な設定はすべてカットオフの中に含まれるので、ETK プロセスを行うにあたって DISTAB メソッドについての詳しい知識は必要はありませんが、ETK メモリを診断する際に役立つように、DISTAB メソッドの動作原理をここで簡単にご紹介します。

ECU とシミュレーションボードの間のデータ転送のための通信メカニズムは「バイパスチャンネル」と呼ばれます。

ES1201 または ES1231 を使用する場合、ECU 上には 2 つのバイパスチャンネルが設けられ、そのうちの 1 つ（チャンネル A、高優先度）は ECU の角度同期グリッドで実行され、もう 1 つ（チャンネル B、低優先度）は時間同期グリッドで実行されます。各バイパスチャンネルごとに、複数のデータバッファのアドレス情報とサイズによって定義されています。ES1232 の場合は、32 のチャンネル（16 のバイパスチャンネルと 16 の測定チャンネル）があり、各チャンネルの名前と優先度は自動的に決められます（210 ページの 10.8.7 項を参照してください）

バイパス通信（AML V1.1）

ここでは、ES1231 のバイパスチャンネル A を例に、バイパス通信の内容を詳しく説明します。処理内容はどのチャンネルも同じです。下の表は、ASAM-MCD-2MC ファイル（`*.a2l`、AMLV1.1）で使用される ETK バイパス用パラメータの一覧です。

注記

以下のパラメータ名は、ASAM-MCD-2MC 規格で定義されたものではなく、本項のサンプルリストに補助的なコメントとして付け加えられているものです。

パラメータ	説明
BYPASS_S	バイパス入力データのポインタリストの先頭アドレス（バイパス ch A）
BYPASS_X	バイパス入力データのポインタリストの先頭アドレス（バイパス ch B）
CHNL_S	バイパス入力データのデータバッファの先頭アドレス（バイパス ch A）
CHNL_X	バイパス入力データのデータバッファの先頭アドレス（バイパス ch B）
CHNL_T	バイパス出力データのデータバッファの先頭アドレス（バイパス ch A）
CHNL_Y	バイパス出力データのデータバッファの先頭アドレス（バイパス ch B）
TRGID_S	バイパス入力データのトリガ ID アドレス （バイパス ch A）
TRGID_X	バイパス入力データのトリガ ID アドレス （バイパス ch B）
BPMAX_S	バイパス入力データのデータバッファサイズ （バイパス ch A）
BPMAX_X	バイパス入力データのデータバッファサイズ （バイパス ch B）
BPMAX_T	バイパス出力データのデータバッファサイズ（バイパス ch A）
BPMAX_Y	バイパス出力データのデータバッファサイズ （バイパス ch B）
TRGSEG_A	バイパス ch A のトリガアドレス
TRGSEG_B	バイパス ch B のトリガアドレス

表 9-1　バイパス通信用パラメータ（AML V1.1）

チャンネル A 用のパラメータは、ASAM-MCD-2MC ファイルの `IF_DATA ETK` セクション内の `QP_BLOB` で定義されます。

```
/begin IF_DATA ETK
  /begin SOURCE  "BYPASS A"
   0
   0
   /begin
     QP_BLOB
```

```
      4    /* Acquisition raster;               */
           /* 1=A (typ. angle synchronous)      */
           /* 2=B (typ. time synch. 10ms)       */
           /* 3=C (typ. time synch. 100ms)      */
           /* 4=S/T angle synch. (bypass only)  */
           /* 5=X/Y time synch. (bypass only)   */
      100       /* BPMAX_S                   */
      0x81025E  /* BYPASS_S                  */
      0x3801E0  /* CHNL_S                    */
      0x38302E  /* TRGID_S                   */
      2         /* trigger repetition rate   */
                /*(worst case)               */
      100       /* BPMAX_T                   */
      0x8103F2  /* CHNL_T                    */
    /end QP_BLOB
   /end SOURCE
   ...
/end IF_DATA
```

次の図は、DISTAB データ交換メソッドによるバイパスサイクルのプロセスを、バイパスチャンネル A を例にして図示したものです。図中の番号はバイパスサイクルの各ステップを表します。

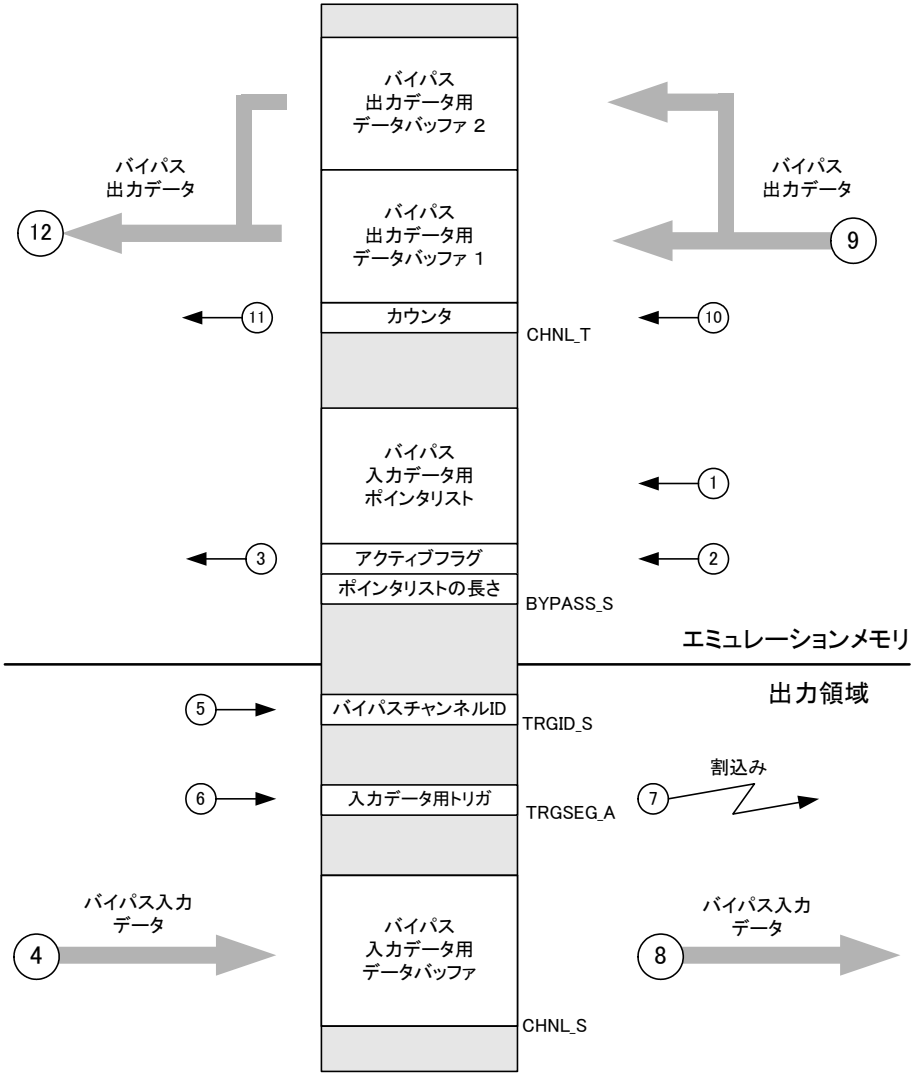


図 9-1　バイパスサイクルの処理の流れ（DISTAB メソッド）

1. シミュレーションボード上でバイパス実験が開始されると、バイパス入力データ用のポインタリストにポインタが書き込まれます。

DISTAB12	DISTAB13
シミュレーションボードに送信される各バイトごとに、対応する ECU メモリのバイトアドレスを指し示すポインタが使用されます。DISTAB 12 メソッドではデータは常にバイト単位で転送されるので、1 ワードを送信するためにはポインタが 2 つ必要です。ポインタリストがすべて設定されると、ポインタリストの長さがリストの先頭バイトに書き込まれます。	ポインタリストの 4 ～ 7 番目のバイトには、8 バイト、4 バイト、2 バイト、および 1 バイトシグナルの数が順に格納され、それ以降のバイトに、8 バイトシグナルから 1 バイトシグナルの順でシグナルアドレスが格納されます。各アドレスは 4 バイトで構成され、アドレスのバイトオーダーは `Byte Order` というパラメータで決定されます（138 ページのリストを参照してください）。

   このステップにおいて、バイパス出力データ用のバイパスオフセット（143 ページのステップ 11. を参照してください）も転送されます。

2. ポインタリストのデータがすべて設定されると、アクティブフラグが 1 にセットされ、これによって ECU とシミュレーションボードとの通信が開始されます。ここまでの処理（ステップ 1 および 2）は、バイパス実験開始時に一度だけ実行されます。

DISTAB12	DISTAB13
アクティブフラグ：ポインタリストの 2 番目のバイトの最後のビット	アクティブフラグ：ポインタリストの先頭バイト（バイト 0）の先頭ビット（ビット 0）

3. ECU は、常にポインタリストのアクティブフラグバイトを調べます。そしてフラグが 1 にセットされると、バイパスがアクティブになり、データ転送が開始されます。
4. ECU が、バイパス入力データを適切なデータバッファ（`CHNL_S` で示されます）に書き込みます。

DISTAB12	DISTAB13
データは、ポインタリスト中の各ポインタが指し示すデータバッファに、1 バイトずつ書き込まれます。リストの先頭バイトで示されるバイト数の送信が終わると、転送が終了します。	データは、シグナル単位で、ポインタリストに示される順、つまり 8 バイトシグナルの 1 番目のシグナルから順に書き込まれます。`Byte Order` というパラメータによって、どのバイトから書き込まれるかが決まります。

5. ECU は現在のバイパスチャンネルの ID を TRGID_S で示されるアドレスに書き込みます。全部で 5 つあるチャンネルのうちチャンネル ID として使用できるアドレスは 2 つだけで、残りの 3 つのチャンネルは適合に使用されます。
6. ECU により、TRGSEG_A で示されるトリガアドレスにランダムな値がロードされます。
7. このトリガアドレスへの書き込みによって、シミュレーションプロセッサへの割り込みが発生します。
8. シミュレーションプロセッサはチャンネル ID を TRGID_S で示されるアドレスから読みとって、どのチャンネルが初期化されたかを判断し、対応するデータバッファの内容を読み取ります。そして ASAM-MCD-2MC ファイル内の変換式などの情報をもとに、ECU から得られたデータを物理モデル用の物理値に変換し、ASCET モデルで処理できるようにします。
9. シミュレーションプロセッサは、バイパスされたファンクションをバイパス入力データに基づいて実行します。実行結果はバイパス出力データとして、適切なデータバッファに書き戻されます。データバッファは各バイパスチャンネル用に 2 つあり、それらが交互に使用されます。まず、シミュレーションプロセッサが内部的にカウンタをインクリメントし、それが後に CHNL_T で示されるアドレスに書き込まれます。そしてこのカウンタは上記 7 の割り込みが発生するとインクリメントされ、カウンタの値に応じて、2 つのデータバッファのどちらが書き込み対象になるかが決まります。つまり、カウンタの値が奇数なら 1 番目のデータバッファが使われ、偶数なら 2 番目のデータバッファが使われます。
10. データバッファへの書き込みが終わると、内部カウンタが CHNL_T で示されるアドレスに書き込まれます。
11. ECU がカウンタ値を読み取り、その値によってどちらのデータバッファを読み取るかが決まります。値が奇数なら 1 番目のデータバッファが、また偶数なら 2 番目のデータバッファが読み取り対象になります。これによってバイパス出力データの整合性が保証されます。

    これら 2 つのバッファのレイアウトは、バイパス出力シグナルの数や内容によって変わります。このため、ECU は、各シグナル値がバッファ内のどこに格納されているかを把握するために、バッファの先頭アドレスからの相対位置を示す「バイパスオフセット」を使用します。安全のため、バイパス出力シグナル用のポインタリスト、というものは存在せず、代わりに、ECU ソフトウェア内に定義された、各シグナルのバイパスオフセット用の特殊なパラメータが使用されます。この「バイパスオフセットパラメータ」には、ステップ 1（142 ページ）においてバイパスオフセットがセットされます。これらのパラメータは、ETK の RAM 上の ECU 適合データ内に配置されるため、直接アクセスが可能です。
12. ECU がバイパス出力データを読み取ります。一般に、ECU には安全機構が組み込まれていて、バイパス通信で障害が発生した場合に重大なトラブルが起こらないような対策が施されています。詳細は、ECU のプログラムの内容をご確認ください。

2 つのデータバッファが必要なのは、バイパス出力データを書き戻す処理が同期していないためです。シミュレーションプロセッサの処理結果は、処理が終わった後に直ちに書き戻されますが、データがまだ書き戻されていない時、またはデータが書き戻されている途中に ECU が結果を要求する可能性があります。そこで、バイパス出力データの整合性を保証するために、新しいデータが完全に書き戻されるまで、そのデータのコピーが必ずデータバッファに保持されています。

AML V1.1 以外のバージョン

AML V1.1.1 の QP_BLOB の内容は、AML V1.1 の例と同じです（139 ページを参照してください）。

ただし AML V1.2 には若干の変更点があります。いくつかのパラメータが削除されて新しいパラメータが追加され、そのためパラメータ名（オプション）も変更されています。AML V1.2 の QP_BLOB の内容は以下のようになります。

```
/begin IF_DATA ETK
  /begin SOURCE  "BYPASS A4"
   0
   0
   /begin
     QP_BLOB
      0x0100       /* version 1.0                  */
      15           /* hardware trigger             */
      INDIRECT     /* indirect/direct triggering   */
      5            /* raster number/priority       */
                   /* (32..1)                      */
      BYPASS       /* raster type                  */
                   /*(BYPASS/MEASUREMENT)          */
      0x38302E     /* trigger id/flag address for  */
                   /* indirect triggering          */
      100          /* Max. length of display table */
                   /* in bytes                     */
      0x81025E     /* address of the display table */
                   /* in the memory                */
      0x3801E0     /* address, where the ECU       */
                   /* writes the display values    */
      100          /* Max. size of bypass receive  */
                   /* table                        */
      0x0          /* StartAddress of the Adress   */
                   /* table for bypass output      */
```

```
        0x8103F2    /* Output address of the        */
                    /* bypass table                 */
        2           /* worst case raster timing     */
      /end QP_BLOB
    /end SOURCE
    ...
  /end IF_DATA
```

AML V1.3、AML V1.4、AML V1.6 も使用できます。これらのバージョンと AML V1.2 との相違点は、ASCET-RP で作業する際には何も影響はありません。

注記

AML V1.1.x では、ETK データは 3+2 測定ラスタ用に記述され、これは ES1232[ETK-CTRL-BAS]、ES1231、ES1200/1 に対して有効です。また **AML V1.2 以降**のバージョンでは ETK データはマルチラスタ用に記述され、これは ES1232[ETK-CTRL-ADV] に対してのみ有効です。

9.6 ETK バイパスプロジェクトに必要な情報とデータ

ETK バイパスプロジェクトの作成を始めるには、ECU についての情報やデータが必要です。これらは、一般的に、バイパスフックを実装した ECU プログラマから入手できます。本項に示す項目は、情報を確実かつ円滑に得るためのチェックリストとしてお役立てください。

ECU プログラム

バイパス処理部分にバイパスフックプログラムがロードされている必要があります。

ASAM-MCD-2MC ファイル

ASAM-MCD-2MC ファイルは、ECU のデータ配置の記述などの目的で使用されます。ETK バイパスパッケージは、ASAM-MCD-2MC ファイルを使用して入力／出力変数にアクセスするため、ECU の現在のソフトウェアバージョンに対応する ASAM-MCD-2MC ファイルが必要です。使用できる AML のバージョンは、ハードウェアによって異なります。詳しくは、各ボードのドキュメントを参照してください。

ECU プロセッサのデータフォーマット

プロセッサファミリが異なれば、ワードデータの格納フォーマット（*Big Endian / Little Endian*）も異なります。ETK バイパスをセットアップする際には、ECU のプロセッサが *Little Endian* と *Big Endian* のどちらのフォーマットあるか、また、そのワードデータ格納フォーマットが ASAM-MCD-2MC ファイルに記述されていて HWC エディタで読み取ることができるかどうかを確認しておく必要があります。

Motorola 社のプロセッサなどで使用されている *Big Endian* フォーマットの場合は 1 番目のバイトが*最上位バイト* で、Intel 社のプロセッサなどで使用されている *Little Endian* フォーマットでは、1 番目のバイトが最下位バイトになります。

ベースアドレス

転送チャンネルの先頭アドレスとトリガアドレスが ETK バイパスパッケージで定義されているか、あるいは先頭アドレスが ASAM-MCD-2MC ファイルに定義されていてそれを HWC エディタで読むことができなければなりません。各ベースアドレス（先頭アドレス）については、137 ページの「ECU と実験システム間のデータ交換」を参照してください。

バイパスを起動するためのパラメータ

バイパスを起動するには、ECU プログラム内でバイパスをイネーブルにする必要があります。バイパスがイネーブルのときは、ECU プログラムの処理結果の代わりに、シミュレーションボードでの処理結果が使用されます。個々のバイパスファンクションは、INCA でのパラメータ設定により有効になります。

バイパス出力変数

ETK バイパスプロジェクトでは ECU 内の一部の変数しか変更できません。これらの変数、いわゆるバイパス出力変数は、ECU ソフトウェアのプログラマがバイパスフック内に定義します。バイパスプロジェクトを作成する際は、バイパス出力変数がわかっているか、あるいはそれらの変数に関する情報が ASAM-MCD-2MC ファイルに格納されていて、それを HWC エディタで読みとることができなければなりません。

near 領域での変換に用いるビットマスク

DISTAB12 メソッドでは、ECU とシミュレーションボードの間のデータ転送において、16 ビットメモリアドレスしか扱われません。ECU のメモリアドレスが 16 ビットアドレスより大きい場合には、データページポインタを使用してアドレスを 16 ビットアドレスに変換する必要があります。これには `longAdrANDMask` および
`longAdrORMask` というビットマスクを使用し、これらのビットマスクは HWC エディタで指定します。

エラー時の対処／安全機構

ECU には通常、バイパス通信で障害が発生した場合の ECU の動作を定義した、安全機構が組み込まれています。これによって、バイパスファンクションの代わりに ECU ファンクションの結果が使用されたり、あるいは ECU を *Reset* または *Emergency* モードに切り替える、などの処理が行われます。特に実車でのバイパス実験の場合には、障害発生時の ECU の動作を知っておくことが重要です。

10 HWC アイテム

本章では、ASCET で使用できるすべての HWC アイテムについて説明します。

10.1 実装されているアイテム

HWC エディタで **Options → Show Installed Items** を選択すると、RTIO パッケージに実装されているアイテムが表示されます。

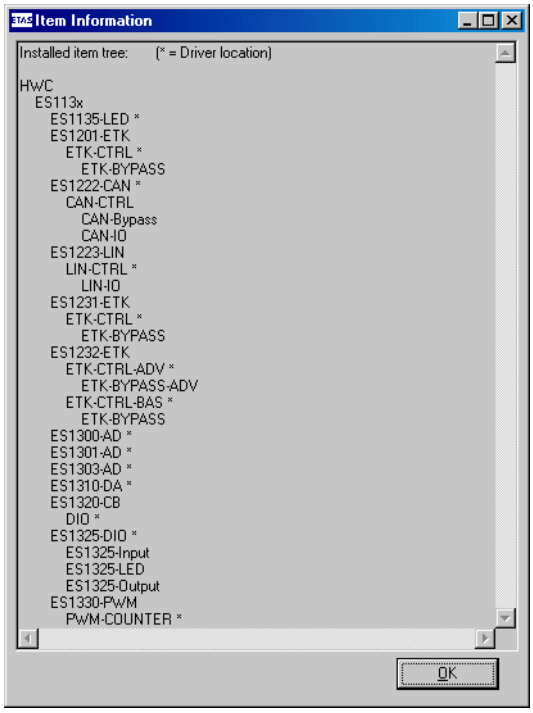


図 10-1 “Item Information”ダイアログボックス（* = HW ドライバ）

以降の項では、各アイテムごとに表示される HWC エディタタブを紹介します。ここではアイテム固有のオプションについてのみ説明されているので、すべてのアイテムで共通に使用できるオプションについては、118 ページ「コンフィギュレーションタブ」の項を参照してください。特に “Mappings” タブについては、アイテム固有のオプションがないため、「コンフィギュレーションタブ」の項でのみ説明されています。

10.2 ES1135-LED

この項では、シミュレーションボード ES1135 のフロントパネル上に設置された LED についての設定を説明します。

10.2.1 Globals（ES1135-LED デバイス）

“Globals”タブで、LED デバイスの一般的なオプション設定を行います。

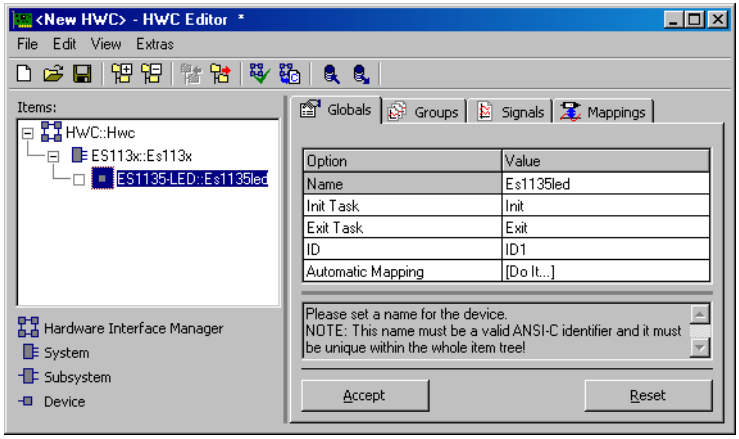


図 10-2 ES1135-LED デバイスの“Globals”タブ

Automatic Mapping

このオプションで、シグナルと ASCET メッセージの間の自動割り当てを行います。詳しい情報は、173 ページの「Automatic Mapping」の項を参照してください。

10.2.2 Groups（ES1135-LED デバイス）

“Groups”タブでは、ES1135-LED デバイスの各シグナルグループのオプション設定を行います。

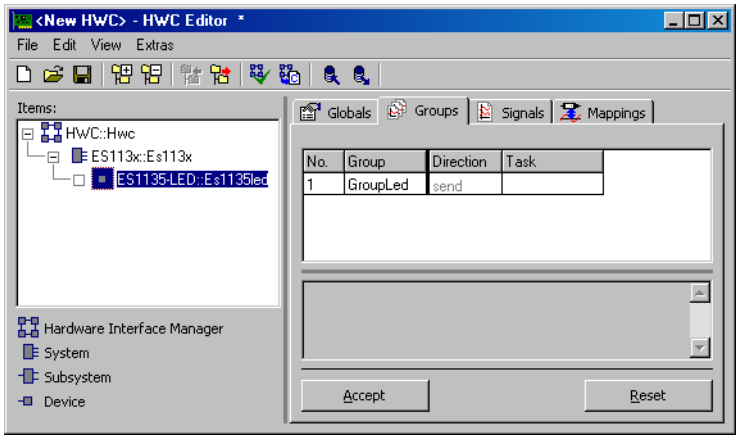


図 10-3 ES1135-LED デバイスの“Groups”タブ

ES1135-LED デバイスの“Globals”タブには、このデバイス固有の項目はありません。一般的な設定内容は、123 ページの 7.4.3 項に説明されています。

10.2.3 Signals（ES1135-LED デバイス）

“Signals”タブでは、ES1135-LED デバイスの各シグナルのオプション設定を行います。

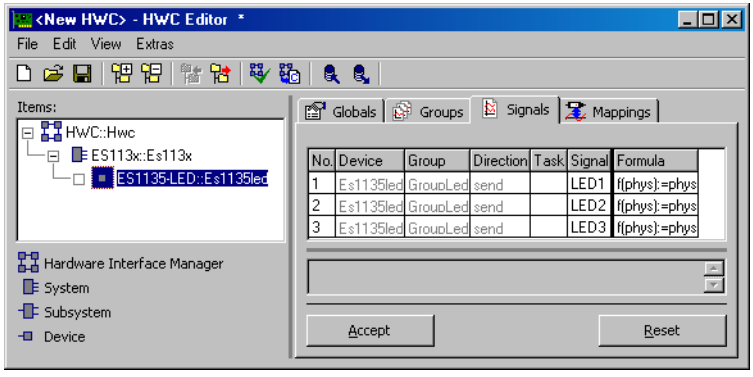


図 10-4 ES1135-LED デバイスの“Signals”タブ

ES1135-LED デバイスの“Signals”タブには、このデバイス固有の項目はありません。一般的な設定内容は、124 ページの 7.4.4 項に説明されています。

10.2.4 Mappings（ES1135-LED デバイス）

このタブ上の設定は、すべてのデバイスについて共通です。125 ページの 7.4.5 項を参照してください。

10.3 ES1201-ETK

ETK インターフェースボード ES1200 または ES1201 を使えば、ETK を搭載した ECU に実験システムを接続することができます。外部からの干渉に強いシリアルインターフェース（8MBit/s）で、ECU メモリの内容を ETAS 実験システムに転送したり修正したりすることができます。

ES1200 は 1 基、ES1201 は 2 基の ETK インターフェースを搭載しています。それ以外の点では、これらのボードの機能はまったく同じです。

注記

ES1201 ボードを使用して ETK と通信できるようにするためには、特別なシステムサービスが必要です。これらのサービスは、ES1111（VCU）および ES1120（VCU2）システムコントローラボードにしか搭載されていません（詳細は、67 ページの「ハードウェア - ES1000.x 実験システム」という項を参照してください）。

10.3.1 Globals（ES1201-ETK サブシステム）

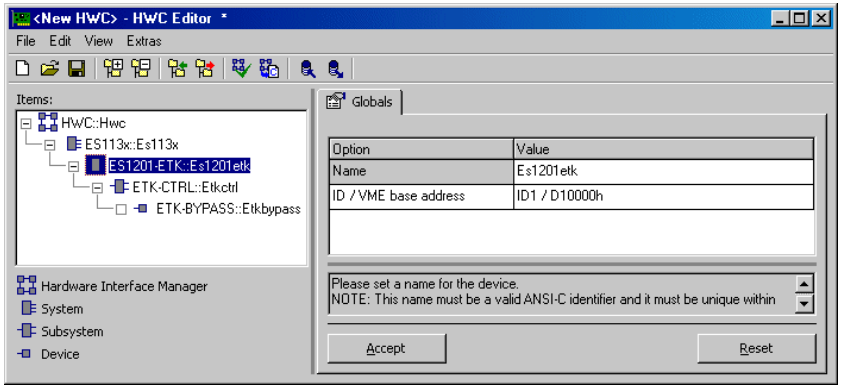


図 10-5 ES1201-ETK サブシステムの“Globals”タブ

ID / VME base address

この行では、VME ベースアドレスの設定を行います。この設定は ES1201 ボード上のスイッチ設定と一致している必要があります。ES1201 の場合、2 種類の異なる VME ベースアドレスから選択することができ、1 つの ETAS 実験システムで最大 2 枚の ES1201 を使用することができます。

ES1201-ETK サブシステムには最大 2 つの ETK-CTRL サブシステムを割り当てることができます。これらの ETK-CTRL サブシステムは、ボード上の 2 基の ETK インターフェースコントローラに相当します。

ES1201-ETK サブシステムは、ES1200 を組み込むためにも使用できます。このボードには ETK インターフェースが 1 基しかないので、ETK-CTRL サブシステムを 1 つしか割り当てることができません（ETK ポート A）。

10.3.2 Globals（ETK-CTRL サブシステム）

ETK-CTRL サブシステムの“Globals”タブで、実際の ETK コントローラまたは ETK ポートを ETK-CTRL サブシステムに割り当てます。

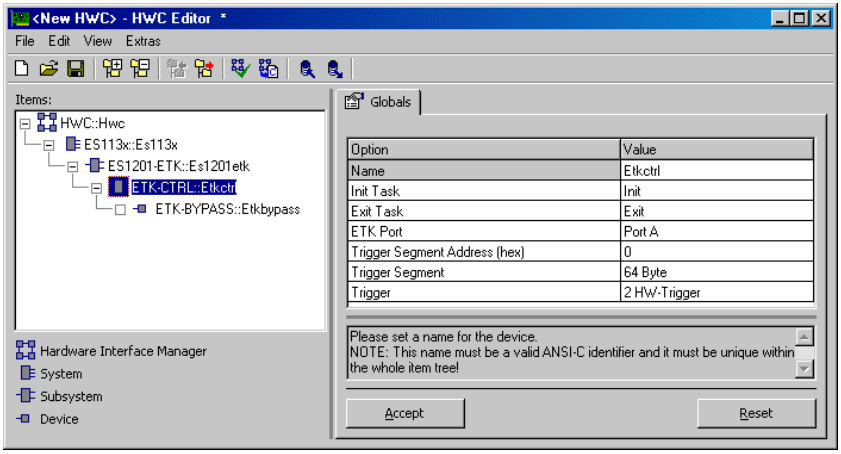


図 10-6 ETK-CTRL サブシステムの“Globals”タブ

ETK Port

独立した 2 つの ETK チャンネル（ポート A、ポート B）のうちの 1 つ、つまり ES1201 の 2 基の ETK コントローラのうちの 1 つを、ETK インターフェースとして割り当てます。ES1200 の場合には、ポート A だけを選択できます。

Trigger Segment Address (hex)

DISTAB 処理のトリガセグメントアドレスを指定します。

ETK バイパスアイテムとそれに関連する ASAM-MCD-2MC プロジェクトが ETK-CTRL サブシステムに割り当てられている場合には、この設定を編集することはできません。関連する変数がその ASAM-MCD-2MC プロジェクト内に正しく定義されている場合には、この設定は自動的に設定されます。

トリガセグメントアドレスは、バイパスチャンネル経由のデータ転送を制御するために使用されるハードウェアトリガアドレスの位置を規定します。トリガセグメントアドレスに対するハードウェアトリガアドレスの相対位置は、DISTAB データ交換プロセス内で固定されています。

64 バイトのトリガセグメントの先頭アドレス（8 ビット幅または 16 ビット幅のバスアクセスの場合）は、64 で割り切れる偶数のアドレスでなければなりません。また、128 バイトのトリガセグメントの先頭アドレス（32 ビット幅のバスアクセスの場合）は、128 で割り切れるアドレスでなければなりません。

トリガセグメントのフォーマットは、DISTAB 12 および DISTAB 13 インターフェースに定義されています。

Trigger Segment

トリガセグメントのサイズを定義します。

ETK バイパスアイテムとそれに対応する ASAM-MCD-2MC プロジェクトが ETK-CTRL サブシステムに割り当てられている場合には、この設定を任意に編集することはできません。対応する変数がその ASAM-MCD-2MC プロジェクト内に正しく定義されていると、この値は自動的に設定されます。

トリガセグメントのアドレススペースを ECU で使用することはできません。16 ビット幅のトリガアドレスの場合にはトリガセグメントの大きさは 64 バイトで、32 ビット幅のトリガアドレスの場合にはトリガセグメントの大きさは 128 バイトです。

Trigger

ここには、バイパスに使用されるハードウェアトリガアドレスの数（2 または 16）を設定します。

1 つの ETK バイパスアイテムとそれに対応する ASAM-MCD-2MC プロジェクトが ETK-CTRL サブシステムに割り当てられている場合には、この設定を任意に編集することはできません。対応する変数がその ASAM-MCD-2MC プロジェクト内に正しく定義されていると、この値は自動的に設定されます。

10.3.3 Globals（ETK-BYPASS デバイス）

ETK-BYPASS というアイテムの“Globals”タブで、バイパス処理に必要なすべてのパラメータを定義します。バイパス処理を行うには、バイパス機能を持ち ETK が装着されている ECU と、バイパス用のソフトウェアが必要です。

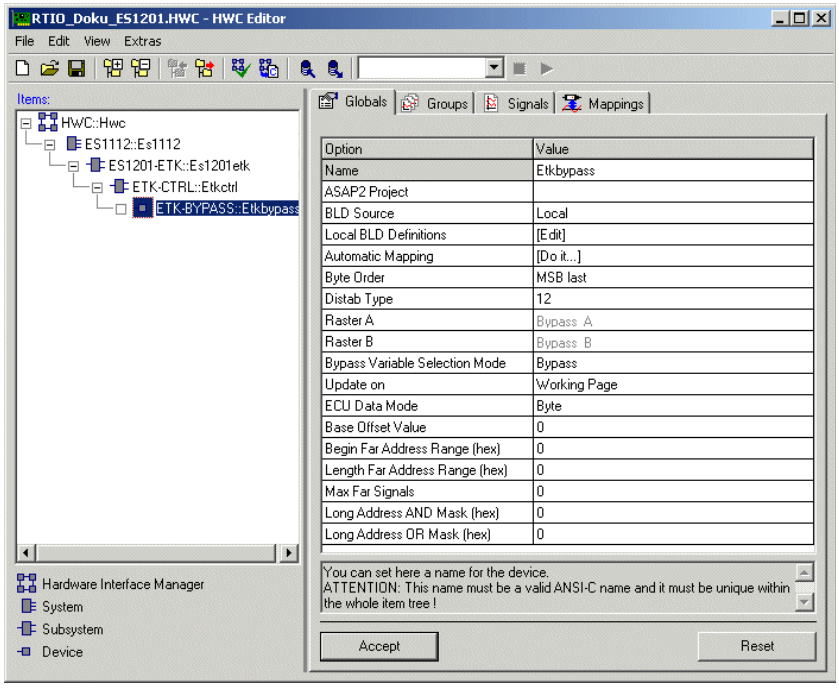


図 10-7 ETK-BYPASS デバイスの“Globals”タブ

ETK バイパスには ASAM-MCD-2MC プロジェクト（AML V1.1）が必要です。プロジェクトは、データベース内に ASAM-MCD-2MC ディスクリプションファイルを読み込んで作成します。

注記

AML V1.2 以降のバージョンを使用した ASAM-MCD-2MC プロジェクトは読み込めません。そのようなプロジェクトをデバイスに割り当てようとすると、次のようなエラーメッセージが出力されます。

```
Error: Incompatible version 0x<version> of QP_BLOB found
in SOURCE '<sName>'. Expected version = 0x1. Selected
ASAM-2MC Project is not suitable for the basic ETK-
Controller (ES1232/ETK-CTRL-BAS, ES1231/ETK-CTRL, ES1201/
ETK-CTRL) ! Please use an advanced ETK-Controller (ES1232/
ETK-CTRL-ADV)!
```

ASAM-MCD-2MC ファイルには、`IF_DATA ETK` というラベル（古いバージョンでは `IF_DATA ASAP1B_ETK`）でバイパスについての設定が定義され、その中に、各バイパスチャンネルの設定を定義する `QP_BLOB` が含まれます。この例は、139 ページの「バイパス通信（AML V1.1）」に記載されています。また `TB_BLOB` には、以下のような一般的な情報が含まれています。

```
/begin TP_BLOB
   0x1000100     /* TP_BLOB version         */
   2             /* Project Base Address    */
   0x0           /* RESET_CFG (only PPC family CPU)*/
   /begin DISTAB_CFG 0xC 0x1 MSB_LAST 0x383000 0x0
    TRG_MOD 0xB7
   /end DISTAB_CFG
   CODE_CHK   /* check whether program and data   */
              /* are matching                     */
      0x0   /* program ID address in data range   */
      0x0   /* program ID length in data range    */
      0x0   /* program ID address in external RAM */
      0x0   /* program ID length in external RAM  */
   ETK_CFG 0xF 0xF0 0xFF 0x3 0xFD 0xEE 0xFF 0x1
                 /* ETK configuration             */
   RESERVED 0x810000     /* start address         */
            0x8103F9     /* length                */
            EXTERN       /* memory attribute      */
            0x-1 0x-1 0x-1 0x-1 0x-1
                         /* mirror offsets 1 - 5  */
/end TP_BLOB
```

ASAM-MCD-2MC ファイルに定義されていない情報は、HWC エディタで設定します。

“ASAM-2MC Project”の右側のセルをクリックすると、ASAM-MCD-2MC プロジェクトを選択するためのダイアログボックスが開きます。

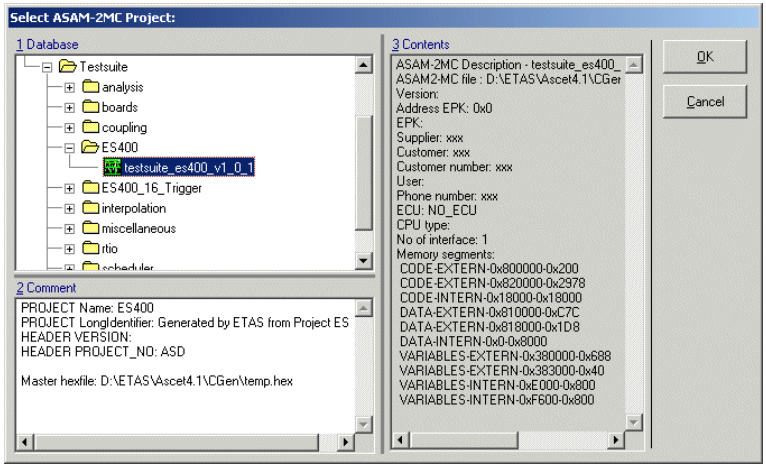


図 10-8 ”Select ASAM-2MC Project”ダイアログボックス

BLD ソース

このオプションは、バイパス出力データとして使用される ASAM-MCD-2MC 変数間の依存関係がどこで定義されるかを決めるものです（BLD = Bypass Label Dependencies）。

以下のいずれかを選択します。

ASAM-2MC File	ASAM-MCD-2MC ファイル内で定義されます。
Local	ETK-BYPASS デバイスに設定されたローカル定義が使用されます（次の「Local BLD Definitions」の項を参照してください）。

Local BLD Definitions

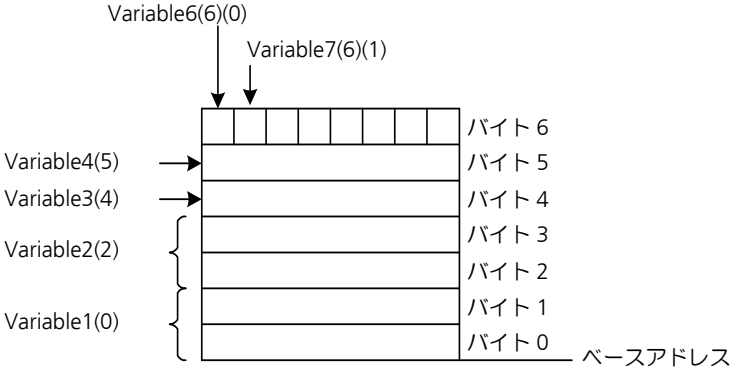
ASCET から ECU へのバイパス出力データの転送は、各ラスタ（角度同期／時間同期）ごとに個別のデータバッファを使用して行われます。

一部の ECU プログラムは、バイパス変数をフレキシブルにこのデータバッファへ割り当てる機能を持っていますが、この機能は、データバッファで転送されるよりも多くのバイパス変数が ECU ソフトウェアによって使用される場合に必要です。

バイパス変数を定義するためには、ECU プログラムに応じて、1 つのバイパス出力変数（Measurement - 測定変数 - として定義されます）ごとに、最大 3 つの Characteristic - 適合変数 - が必要となります。

Byte offset or vector parameter:	バイパス出力変数の位置をデータバッファの先頭からのオフセット（バイト数）で指定します。
Bit offset:	ビット変数にのみ有効で、バイト内のビット位置で指定します。
Source:	一部の ECU は、バイパス変数をバイパスラスタ（角度同期／時間同期）に自由に割り当てる機能を持っています。このパラメータに、割り当て先を指定する値が格納されます（0 = 角度同期、1 = 時間同期）。

以下に例を示します。



上の例で、ビット変数 `Variable6` は、バイパス出力変数用データバッファのバイト 6 のビット 0 に格納されているので、オフセット値は「バイトオフセット：6 ／ビットオフセット：0」です。

もう 1 つのビット変数である `Variable7` はバイト 6 のビット 1 に格納されているので、オフセット値は「バイトオフセット：6 ／ビットオフセット：1」です。

`Variable3` は 1 バイトで、バッファ内のバイト 4 に格納されているため、そのバイトオフセットは 4 で、また `Variable4` のバイトオフセットは 5 です。

`Variable1` はデータバッファの先頭にあるので、そのバイトオフセットは 0 です。この `Variable1` は 2 バイトを占めるワードデータなので、`Variable2` のバイトオフセットは 2 になります。

“Value” 列の `[Edit]` という項目をクリックしてテキストエディタを開き、バイパスラベル間の依存関係のローカル定義を設定します（BLD Bypass Label Dependencies）。

テキストは複数の行で構成され、各行に 1 つのバイパス出力シグナルを記述します。バイパス出力シグナルのラベルの依存関係は、以下のような構文で定義します。

```
<Grid>, <Signal>, [<Byte Offset>], [<Bit Offset>], [<Source>]
```

`Grid:`	“A”（角度同期）、“B”（時間同期）、“AB”（角度および時間同期）
`Signal:`	バイパス出力変数に割り当てられる ASAM-MCD-2MC の Measurement の名前
`Byte Offset:`	バイトオフセットを指定する ASAM-MCD-2MC の Characteristic の名前
`Bit Offset:`	ビット変数のビット位置を指定する ASAM-MCD-2MC の Characteristic の名前
`Source:`	2 つ以上のラスタ（AB）がある場合、現在選択されているラスタを指定するための ASAM-MCD-2MD の Characteristic の名前
`//`	コメント行であることを示します。

ここで定義されるパラメータのオフセットは、ASCET によって自動的に設定されます。

ローカル BLD 定義の例：

```
1                   // current version of the BLD
sgetas2             // (optional) name of the dependent
                    // ASAP2 file
                    // (is ignored by BLD reader)
A, t1b_8, EBOTT1_8 // definition of a bypass signal
                    // available for angle sync sample
                    // grid
B, B_t1b, EBOTBT1, BOBBT1
                    // definition of a bypass bit
                    // signal available for time
                    // sync grid
AB, B_Test, B_Test_Vector,, B_Test_Channel
                    // definition of a bypass signal
                    // available for both sample grids
                    // with definition for the actual
                    // sample grid (source) parameter
```

```
A, log_uint8_0_A, log_uint8_0_offset_A.Model_Byp_A,
   log_uint8_0_bitOffset_A.Model_Byp_A
                  // definition of a bypass signal
                  // available for angle sync sample
                  // grid
```

この例は、ASAM-MCD-2MC ファイル（AML V1.1）内に定義された以下のような `Measurement` セクションに対応するものです。

```
/begin MEASUREMENT log_uint8_0_A
   ""
   UBYTE
   ident
   1
   100
   0.0
   1.0
   READ_ONLY
   BIT_MASK 0x8
   ECU_ADDRESS 0xFD00
/end MEASUREMENT
```

Automatic Mapping

詳しい情報は、173 ページの「Automatic Mapping」の項を参照してください。

Byte Order

ECU のプロセッサのワードデータ格納フォーマット（MSB first / MSB last）を表します。この設定は、ASAM-MCD-2MC プロジェクト（138 ページ参照）から読み込まれます。

一般的なバイトオーダーは以下のとおりです。

MSB first	big endian	モトローラ
MSB last	little endian	インテル

Distab Type

ECU とのデータ交換に使用される DISTAB バージョンを表示します。この設定は、ASAM-MCD-2MC プロジェクト（138 ページ参照）から読み込まれます。

DISTAB 12 は、最大 2 バイトのシグナルをサポートします。

DISTAB 13 は 1、2、4、8 バイトのシグナルをサポートします。

Grid A / B

ASAM-MCD-2MC プロジェクトに定義されている 2 つのラスタ、A および B（角度同期と時間同期）を表示します。

Bypass Variable Selection Mode

バイパス出力変数（“Gourp”タブの“send”シグナルグループ）の選択リストに、ASAM-MCD-2MD デスクリプションに“Measurement”として定義されているすべての ECU 変数を表示する（“all”）か、それともバイパス出力変数だけを表示する（“bypass”）かを選択します。

この設定が影響しないバイパス入力変数（“Gourp”タブの“receive”シグナルグループ）については、常に ASAM-MCD-2MD デスクリプションに定義されているすべての測定変数が表示されます。

Update on

バイパス通信において、シミュレーションボードと ECU 間のデータ転送を、ETK のどのページ（`Working Page`、`Reference Page`、`Working & Reference Page`）で行うかを選択します。

ECU Data Mode

ECU プロセッサがデータメモリにアクセスする際に使用するアクセスモード（`byte` アクセス／ `word` アクセス）を指定します。この設定は、ASAM-MCD-2MC プロジェクト（138 ページ参照）から読み込まれます。

Base Offset Value

各バイパス出力変数のバイトオフセット (0、2、または 8 バイト) を指定します。

バイトオフセットのデフォルト値は DISTAB 12 では 0、DISTAB 13 では 8 です。それ以外の設定を使用するのは特殊な場合に限られますので、ECU プログラマとの調整作業が必要になります。

Begin Far Address Range (hex)

FAR アドレス領域の先頭アドレスを指定します。通常、この設定は DISTAB12 を使用する場合にのみ必要です。

DISTAB 12 を使用する C16x マイクロコントローラ搭載の ECU では、ECU の FAR アドレス域の変数を読み書きすることはできません。FAR アドレス域の値はすべて個別に ETK アクセスで読み取られるため、データ転送の速度が低下します。

Length Far Address Range (hex)

FAR アドレス域のサイズを指定します。通常、この設定は DISTAB12 を使用する場合にのみ必要です。

Max Far Signals

FAR アドレス域から読みとることのできる変数の最大数を指定します。通常、この設定は DISTAB12 を使用する場合にのみ必要です。

FAR 領域から多くの変数を読みとることは、ETK バイパスのパフォーマンスに悪影響を与えます。そのため、この範囲から読みとる変数の数は最小限に抑えるようにしてください。

Long Address AND / OR Mask (hex)

この 2 行に、次の“Adress Mapping”オプションで選択されたマスクが表示されます。

Address Mapping

この行は、DISTAB12 が使用されている場合にのみ変更できます。

ここでは、アドレス変換を行う際のビットマスクを選択します。16 ビットよりも長いメモリアドレスを使用する C16x マイクロコントローラ搭載の ECU の場合、このビットマスクを使用して、DISTAB 12 が使用可能な 16 ビットアドレスに変換することができます。

式：

```
<ECU_address> :=
        <Memory_address> & <AND Mask> | <OR Mask>
```

ここで選択されたマスクの値は、“Long Address AND / OR Mask (hex)”の行に表示されます。以下に、“Address Mapping”の設定に応じたマスクの値を示します。

Address Mapping	**Long Address**	
	AND Mask (hex)	**OR Mask (hex)**
Mask 16 Bit	0xFFFF	0x0000
C167x DPP0	0x3FFF	0x0000
C167x DPP1	0x3FFF	0x4000
C167x DPP2	0x3FFF	0x8000
C167x DPP3	0x3FFF	0xC000

10.3.4 Groups（ETK-BYPASS デバイス）

ETK-BYPASS デバイスのシグナルグループの数は、4 に固定されています。最初の 2 つのグループが 1 番目のバイパスラスタ（通常は角度同期）に割り当てられ、残りの 2 つのグループが 2 番目のバイパスラスタ（通常は時間同期）に割り当てられます。

各ラスタは 2 つのシグナルグループの送受信を行います。

受信シグナルグループのデータは標準的な DISTAB メソッドで転送され、送信シグナルグループのデータは ECU と RTIO の間でバイパステーブルメソッドによって交換されます。」

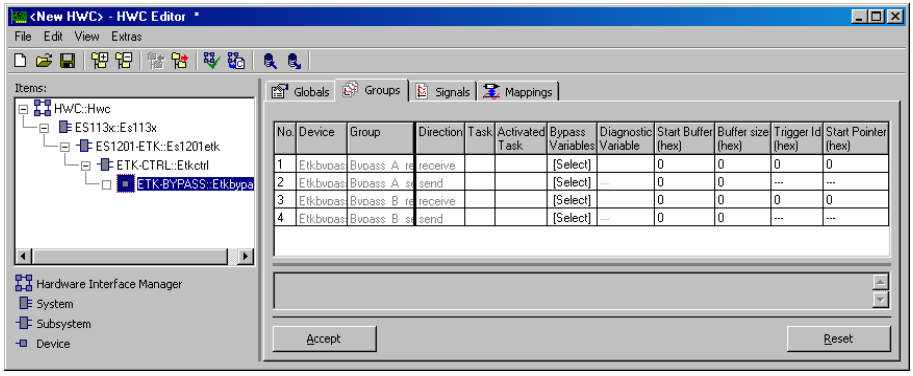


図 10-9 ETK-BYPASS デバイスの“Groups”タブ

注記

グループ名、シグナル名、シグナル方向を変更した際、それ以前にマッピングされていた ASCET メッセージは自動的に再割り当てされないため、マニュアル操作で再度マッピングを行う必要があります。

Task

バイパスファンクションが正しく実行されるためには、以下のようにタスクが割り当てられている必要があります。

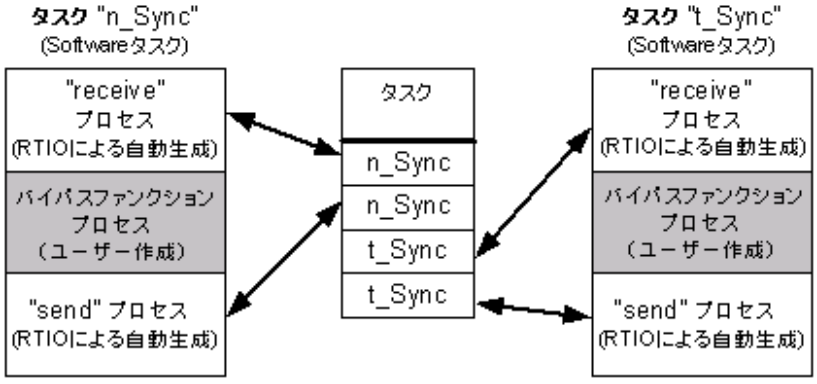


図 10-10 ETK バイパスを行なう際の標準的なタスク割り当て

ActivatedTask

このフィールドに software タスクが指定されていると、データ送受信時に、明示的な“activate Task”コールによってそれらが起動されます。

Bypass Variables

10.5.4 項の「Bypass Variables」（186 ページ）を参照してください。

Diagnostic Variable

10.5.4 項の「Diagnostic Variable」（187 ページ）を参照してください。

Start Buffer (hex)

バイパス入力データとバイパス出力データのデータバッファの先頭アドレスを指定します。

バイパス処理に対応した ASAM-MCD-2MC プロジェクトには、通常、これらのアドレスが含まれています。その場合、このフィールドには ASAM-MCD-2MC プロジェクトに定義された値が表示され、編集はできません。

この項目に関連して、以下の値が ASAM-MCD-2MC プロジェクトに定義されています（表 9-1 の“Parameter”カラムを参照してください）。

`CHNL_S`、`CHNL_X`、`CHNL_T`、`CHNL_Y`

Buffer size (hex)

バイパス入力データとバイパス出力データのデータバッファのサイズを、バイト単位で指定します。

バイパス処理に対応した ASAM-MCD-2MC プロジェクトには、通常これらの設定が含まれています。その場合、このフィールドには ASAM-MCD-2MC プロジェクトに定義された値が表示され、編集はできません。

この項目に関連して、以下の値が ASAM-MCD-2MC プロジェクトに定義されています（表 9-1 の“Parameter”カラムを参照してください）。

`BPMAX_S`、`BPMAX_X`、`BPMAX_T`、`BPMAX_Y`

Trigger Id (hex)

“receive”シグナルグループについて、DISTAB 用のトリガ識別子のアドレスを指定します。

バイパス処理に対応した ASAM-MCD-2MC プロジェクトには、通常これらの設定が含まれています。その場合、このフィールドには ASAM-MCD-2MC プロジェクトに定義された値が表示され、編集はできません。

この項目に関連して、以下の値が ASAM-MCD-2MC プロジェクトに定義されています（表 9-1 の“Parameter”カラムを参照してください）。

`TRGID_S`、`---`、`TRGID_X`、`---`

Start Pointer (hex)

“receive”シグナルグループについて、DISTAB 用のポインタリストのアドレスを指定します。

バイパス処理に対応した ASAM-MCD-2MC プロジェクトには、通常これらの設定が含まれています。その場合、このフィールドには ASAM-MCD-2MC プロジェクトに定義された値が表示され、編集はできません。

この項目に関連して、以下の値が ASAM-MCD-2MC プロジェクトに定義されています（表 9-1 の“Parameter”カラムを参照してください）。

BYPASS_S、---、BYPASS_X、---

10.3.5 Signals（ETK-BYPASS デバイス）

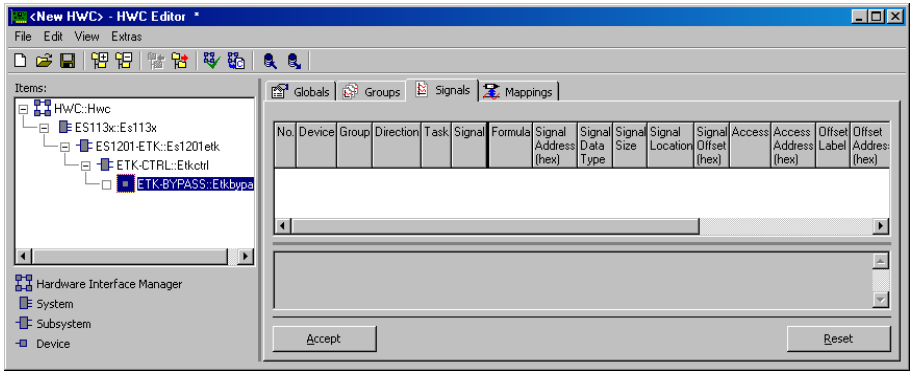


図 10-11 ETK-BYPASS デバイスの“Signals”タブ

注記

“Signals”タブにはバイパス変数のステータス値が表示されます。ユーザーが編集できる項目はありません。

バイパス変数は、シグナルグループごとにバイトサイズ順にソートされます。“send”シグナルの場合は、8-4-2-1 バイトとなります。

“receive”シグナルの場合、そのシグナルが ECU コントローラの内部変数であれば、DISTAB によって間接的に読み込むことしかできず、また ECU コントローラの外部メモリに格納される変数であれば、ETK で直接読み込むこともできます。これを区別するために、各シグナルは、8-4-2-1（直接）と 8-4-2-1（間接）とに分けて表示されます。

Signal Address (hex)

バイパス変数に割り当てられた ASAM-MCD-2MC の Measurement のメモリアドレスメモリアドレスを表示します。

Signal Data Type

バイパス変数の型を表示します。

Signal Size

バイパス変数のデータサイズをバイト数で表示します。

Signal Location

この項目は“receive”シグナルに関してのみ有効で、バイパス変数のメモリロケーション（Internal ／ External）を示します。

- Internal

  このバイパス変数が ECU コントローラの内部 RAM に配置されることを示します。この場合、DISTAB によってのみ読み込み可能です。

- External

  このバイパス変数が ECU コントローラの外部 RAM に配置され、ETK で直接読み込むことも可能であることを示します。

Signal Offset (hex)

バイパス変数の、データバッファ内のインデックス（0 ～）を示します。

Access

この項目は“receive”シグナルに関してのみ有効で、測定変数にアクセスするために使用されるメソッドを表示します。

“Distab”	DISTAB 経由
“Direct”	ETK によるダイレクトアクセス

Access Address (hex)

変換されたバイパス変数のターゲットアドレスが表示されます。

ASAM-MCD-2MC プロジェクトには、ETK アクセスに使用される“memory segment”というエレメントが含まれていて、ここにターゲットアドレスを算出するための変換方法が格納されています。

DISTAB12 を使用する場合は、さらに“Globals”タブのアドレスマスク、“AND Mask”および“OR Mask”が使用されます。

変換式：

```
<target_address> :=
        (<signal_address> & <AND Mask> | <OR Mask>)
```

（“memory segment”変換による）

Offset Label

この項目は“send”シグナルに関してのみ有効で、バイパス変数について BLD 参照が定義されている場合に使用されるベクタパラメータの名前を表示します（詳しい情報は、155 ページの「Local BLD Definitions」を参照してください）。

Offset Address (hex)

この項目は“send”シグナルに関してのみ有効で、バイパス変数について BLD 参照が定義されている場合に使用されるベクタパラメータのアドレスを表示します（詳しい情報は、155 ページの「Local BLD Definitions」を参照してください）。

Offset Value (hex)

この項目は“send”シグナルに関してのみ有効で、バイパス開始時に、必要なバイパス変数を有効にするために必要なパラメータを調整する値を表示します。この値はバイパス変数について BLD 参照が定義されている場合にのみ有功です（詳しい情報は、155 ページの「Local BLD Definitions」を参照してください）。

変換式：

```
Offset Value =
      Signal Offset + Base Offset Value
```

（`Base Offset Value` は“Globals”タブで設定されます。）

Bit Label

この項目は“send”シグナルに関してのみ有効で、ビットデータであるバイパス変数について BLD 参照が定義されている場合に使用される、ビットオフセットパラメータの名前を表示します（詳しい情報は、155 ページの「Local BLD Definitions」を参照してください）。

Bit Address (hex)

この項目は“send”シグナルに関してのみ有効で、ビットデータであるバイパス変数について BLD 参照が定義されている場合に使用される、ビットオフセットパラメータのアドレスを表示します（詳しい情報は、155 ページの「Local BLD Definitions」を参照してください）。

Bit Value

この項目は“send”シグナルに関してのみ有効で、バイパス開始時に、データバイト内のビットポジション（0 ～）を指定するために必要なビットオフセットパラメータを調整する値を表示します。この値はビットデータであるバイパス変数について BLD 参照が定義されている場合にのみ有功です（詳しい情報は、155 ページの「Local BLD Definitions」を参照してください）。

Src Label

この項目は“send”シグナルに関してのみ有効で、“Source”パラメータの名前を表示します。この値はバイパス変数について BLD 参照が定義されている場合にのみ有功です（詳しい情報は、155 ページの「Local BLD Definitions」を参照してください）。

Src Address (hex)

この項目は“send”シグナルに関してのみ有効で、“Source”パラメータのアドレスを表示します。この値はバイパス変数について BLD 参照が定義されている場合にのみ有功です（詳しい情報は、155 ページの「Local BLD Definitions」を参照してください）。

Src Value

この項目は“send”シグナルに関してのみ有効で、バイパス開始時に、ECU にどちらのラスタで各バイパス変数を更新するかを通知するために必要なパラメータ（Src Label）を調整する値を表示します。角度同期ラスタ (A) の場合はこの値は 0 で、時間同期ラスタ（B）の場合は 1 となります。

この値は、バイパス変数について BLD 参照が定義されている場合にのみ有功です（詳しい情報は、155 ページの「Local BLD Definitions」を参照してください）。

10.3.6 Mappings（ETK-BYPASS デバイス）

このタブ上の設定は、すべてのデバイスについて共通です。125 ページの 7.4.5 項を参照してください。

10.4 ES1222-CAN

ES1222 ボードは、VME バスシステムの CAN インターフェースとして使用されます。このボードには 4 つの CAN チャンネルが搭載されていて、各チャンネルごとに Intel 82527 タイプの CAN コントローラが装備されています。

また、このボードでは、受信したメッセージで VME バス割り込みをトリガして、メッセージのポーリングを繰り返し行わないですむようにもできます。各 CAN コントローラは、それぞれ最大 255 の送信メッセージと最大 255 の受信メッセージを処理できます。専用のプロセッサが使用されるので、システムコントローラの負荷は大幅に軽減されます。

注記

FIFO メモリにはチャンネルあたり 82 のメッセージしか格納できないため、同時に 255 のメッセージを送信することはできません。FIFO メモリのオーバーブローを避けるため、メッセージをいくつかのタスクから何回かに分けて FIFO メモリに送り出す必要があります。

本項では、ES1222 ボード（以前は VSIC ボードと呼ばれていました）を RTIO に組み込む方法について説明します。HWC エディタに ES1222 ボードを組み込むには、“ES1222-CAN”というアイテムを選択します。

基本的は、このボードを使用して、定義済みメッセージによる CAN 通信を行うすべてのリモートターミナルを接続することができます。

10.4.1 Globals（ES1222-CAN サブシステム）

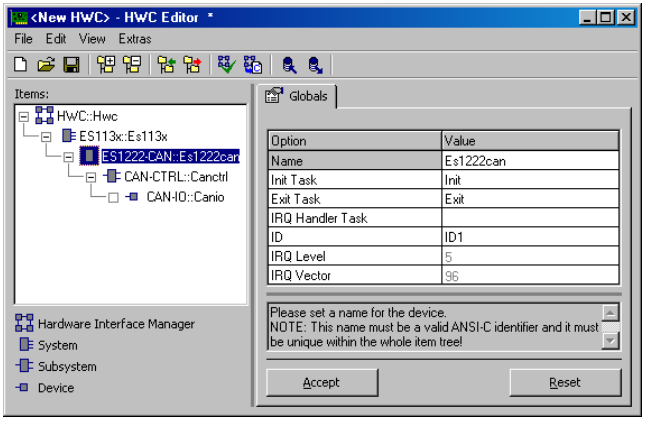


図 10-12 ES1222-CAN サブシステムの“Globals”タブ

ES1222-CAN サブシステムには、最大 4 つの CAN-CTRL サブシステムを割り当てることができます。1 つの“CAN-CTRL”アイテムは、ボード上の 4 基の Intel 82527 CAN コントローラのうちの 1 つに対応します。

IRQ Handler Task

“IRQ Handler Task”は、ES1222 ボードの割り込みを使用する場合に必要です。このタスクは、ASCET プロジェクトエディタのタスクリスト内に、“Software”タスクとして定義されている必要があります。“Max. No. of Activations”フィールドには 2 以上の数（最大 50）を入力してください。

ID

アドレシングするボードのボード番号を入力します。

注記

ES1222 は、システムがオンになったときにハードウェアマネージャにより ID 番号およびフリーアドレス領域が自動的に割り当てられる、“Auto-ID”ボードと呼ばれるタイプのボードです。同じタイプのボードが複数存在する場合には、左から右の順に番号が付けられます。つまり、一番左のボードの番号が 1 になり、その次の番号は 2 になります。1 つの実験システムでは最大 4 枚のボードを使用することができ、4 種類の ID を指定できます。

10.4.2 Globals（CAN-CTRL サブシステム）

CAN-CTRL サブシステムの“Globals”タブでは、実際の 1 つの CAN コントローラまたは CAN コネクタを CAN-CTRL サブシステムに割り当てます。

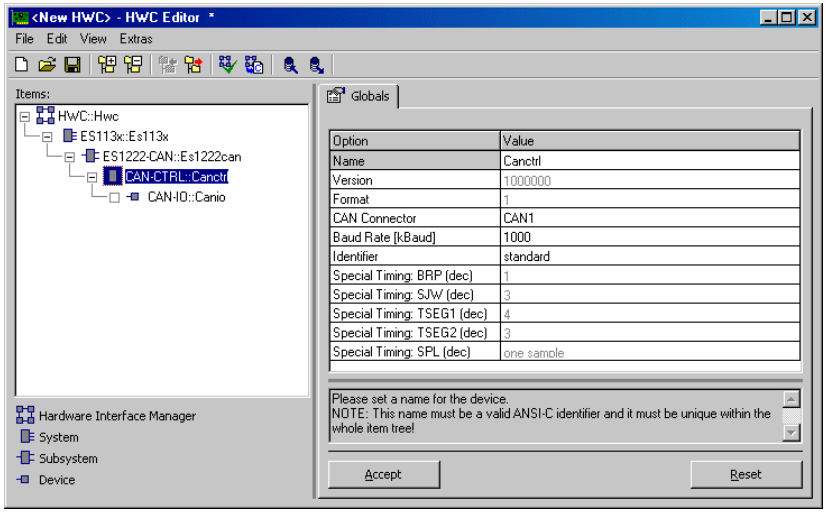


図 10-13 CAN-CTRL サブシステムの“Globals”タブ

CAN Connector

CAN コントローラまたは CAN コネクタ（ポート A、ポート B、ポート C、ポート D）を選択します。

Baud Rate [kBaud]

転送レートを指定します。標準的な 8 種類のボーレート（1000、500、250、125、100、50、20、10 kBaud）から選択できます。また、`<Special Timing>` 設定を選択し、CAN コントローラのビットタイミングおよび転送レートに関して低水準の制御をアクティブにすることもできます。“Special Timing”設定を指定するためには、以下の 5 つのオプションが必要です。

Identifier

CAN メッセージについて、11 ビット識別子を持つ「標準フレーム」と 29 ビット識別子を持つ「拡張フレーム」のどちらを使用するかを選択することができます。*standard* または *extended* を選択します。

注記

CAN-CTRL デバイス内で標準フレームと拡張フレームを混在して使用することはできません。

standard 識別子が選択されていると、CAN-IO デバイスの“Group”タブの“identifer dec/hex”フィールド（176 ページを参照してください）には 11 ビットの値しか入力できません。それより大きな値を入力すると、上位のビットが自動的に削除され、このときワーニングメッセージは**表示されません**。

extended 識別子が選択されていると、同フィールドに 29 ビットの値を入力できます。それより大きな値を入力すると、上位のビットが自動的に削除され、このときもやはりワーニングメッセージは**表示されません**。

Special Timing: BRP (dec)

このオプションはデフォルトでは非表示になっています。“Baud Rate”オプションの値が **<Special Timing>** になっている場合に限り、編集可能です。

このパラメータは、CAN コントローラの入力クロックの転送レートを決定する“Baud Rate Prescaler”を設定するためのものです。この設定値の範囲は、0 ～ 63 です。

この設定についての詳細は、Intel 82527 CAN コントローラのデータシートを参照してください。

Special Timing: SJW (dec)

このオプションはデフォルトでは非表示になっています。“Baud Rate”オプションの値が **<Special Timing>** になっている場合に限り、編集可能です。

このパラメータは“Synchronization Jump Width”を設定するためのものです。この設定値の範囲は、0 ～ 3 です。

この設定についての詳細は、Intel 82527 CAN コントローラのデータシートを参照してください。

Special Timing: TSEG1 (dec)

このオプションはデフォルトでは非表示になっています。“Baud Rate”オプションの値が **<Special Timing>** になっている場合に限り、編集可能です。

このパラメータは“Time Segment 1”を設定するためのもので、サンプリングの間隔を決定するためのタイムセグメントを指定します。この設定値の範囲は、2 ～ 15 です。

この設定についての詳細は、Intel 82527 CAN コントローラのデータシートを参照してください。

Special Timing: TSEG2 (dec)

このオプションはデフォルトでは非表示になっています。“Baud Rate”オプションの値が **<Special Timing>** になっている場合に限り、編集可能です。

このパラメータは“Time Segment 2”を設定するためのもので、サンプリングの間隔を決定するためのタイムセグメントを指定します。この設定値の範囲は、1 ～ 7 です。

この設定についての詳細は、Intel 82527 CAN コントローラのデータシートを参照してください。

Special Timing: SPL (dec)

このオプションはデフォルトでは非表示になっています。“Baud Rate”オプションの値が <Special Timing> になっている場合に限り、編集可能です。

このパラメータは“Sampling Mode”を設定するためのもので、論理ステートを決定するためにシグナルを何回サンプリングするかを決定します。

この設定についての詳細は、Intel 82527 CAN コントローラのデータシートを参照してください。

CAN バス周波数を計算する式は以下のとおりです（Intel 82527 CAN コントローラのデータシートより引用）。

```
CAN bus frequency =
        10 MHz / [(BRP + 1) x (3 + TSEG1 + TSEG2)]
```

10.4.3 Globals（CAN-IO デバイス）

CAN-IO デバイスを使用して、CAN バス上で CAN メッセージの送受信を行うデバイスを簡単にシミュレートすることができます。

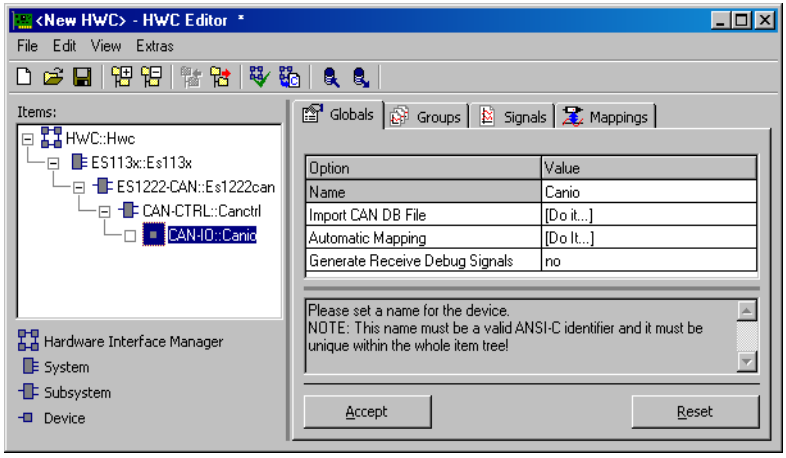
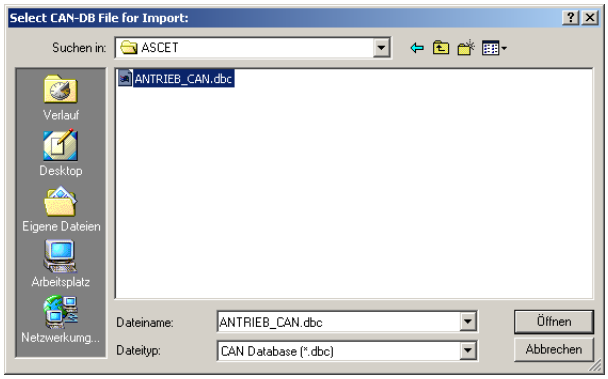


図 10-14 CAN-IO デバイスの“Globals”タブ

Import CAN DB File

このオプションは、Vector Informatik 社製の CANdb データ管理プログラムで作成された CAN データベースファイルを読み込む際に使用します。このファイルを使用して CAN メッセージおよびシグナルを自動的に生成することができます。

[Do it...] をクリックするとダイアログボックスが開きます。



このダイアログボックスで、インポートする CAN-DB ファイルを選択します。

Open ボタンをクリックすると、以下のダイアログボックスが開き、実行する処理を指定することができます。

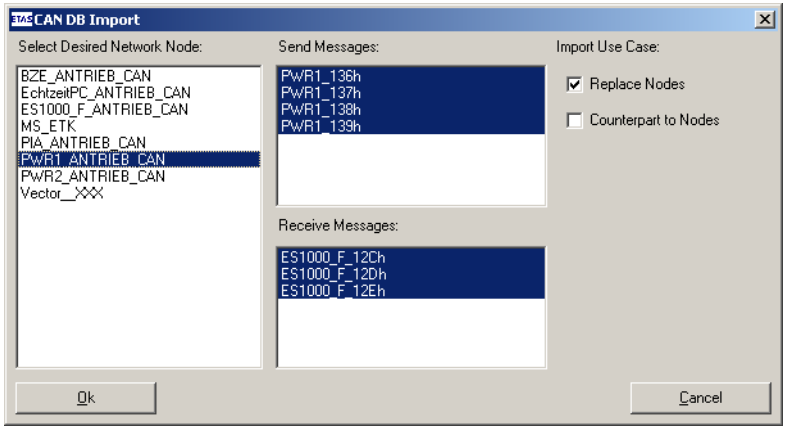


図 10-15 CAN DB Import ダイアログボックス

通常、CAN DB ファイルには、CAN ネットワーク内の複数のノードについての情報が記述されており、“Select Desired Network Node”リストにこれらのノードがすべて表示されます。その右側の 2 つのリスト（“Send Messages”と“Receive Messages”）には、現在選択されているノード用に定義されているすべての CAN メッセージが表示され、ここで選択されたメッセージがインポートされます。また“Import Use Case”の項目で、“Replace Nodes”オプションをオンにすると、CAN-IO デバイスがネットワークノードの役割を負うことになり、たとえばノードの送信メッセージが CAN-IO デバイスの送信メッセージとしても使用されます。

もう 1 つの“Counterpart to Nodes”オプションをオンにすると、CAN-IO デバイスがネットワークノードの相手方になり、たとえば、そのノードの送信メッセージが受信メッセージになります。

OK ボタンで設定内容を確定すると、インポートするデータ（CAN メッセージおよびシグナル）とシグナルグループおよびシグナルがチェックされ、その結果が以下のダイアログボックスに表示されます。

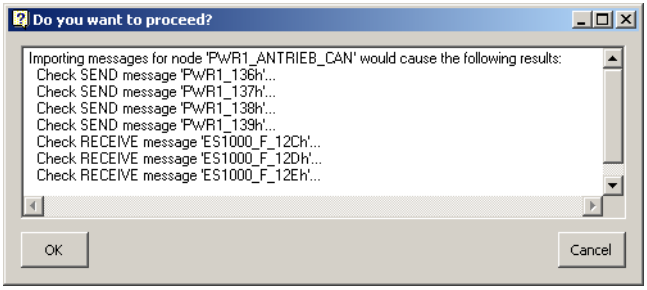


図 10-16“CAN DB Check”ダイアログボックス

この段階では、既存の CAN-IO デバイスはまだ変更されていません。実際のインポート処理はここで“OK”ボタンがクリックされるまでは開始されません。

インポート処理では、インポートされたメッセージがシグナルグループに挿入され、適切なシグナルが確実に定義されるための処理が行われます。

インポートが完了すると、インポート処理の詳細な履歴が、“Monitor”ダイアログボックスに表示されます。

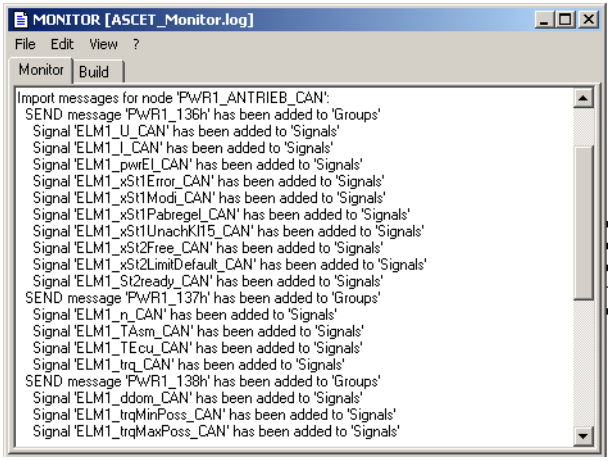


図 10-17“Monitor”ウィンドウに表示されるインポート処理の履歴

インポート時の注意点：

CAN DB ファイルをインポートする際、識別子フィールドには、29 ビット（拡張識別子の場合）または 11 ビット（標準識別子の場合）の識別子が自動的に挿入されます。識別子については 168 ページの「Identifier」を参照してください。

CAN-CTRL アイテムで標準識別子が選択されている場合に、CAN DB ファイルに 29 ビット識別子（ID > 2^{31}）が含まれていると、以下のような処理が行なわれます。

- 11 ビット（ビット [28...18]）のみが識別子フィールドに挿入されます。MSB のビットは無視されます。
- モニタウィンドウにワーニングメッセージが表示されます。

標準または拡張のいずれか一方の識別子しか使用できないため、両方のタイプの識別子が含まれている CAN DB ファイルをインポートすると、何らかの不具合の原因となる場合があります。

Automatic Mapping

このオプションで、シグナルと ASCET メッセージの間の自動割り当てを行います。

`[Do it...]` をクリックするか、または **Extras → Map Item Signals** を選択して、“Automatic Maping”ダイアログボックスを開きます。

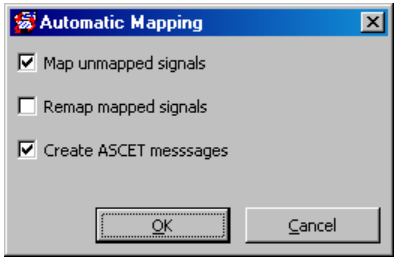


図 10-18 “Automatic Mapping”ダイアログボックスに表示されるマッピングステータス

このダイアログボックスで以下のオプションを設定し、マッピング処理の内容を詳細に指定します。

- **Map unmapped signals**：
  ASCET メッセージにマッピングされていないシグナル（“Mappings”タブ上の“ASCET Message”フィールドが空白になっているシグナル）をすべてマッピングします。つまり、シグナルと同名の ASCET メッセージが検索されてマッピングされます。ただし **Create ASCET message** オプションがオフになっていると、同名の ASCET メッセージが見つからないシグナルはマッピングされません。
- **Remap mapped signals**：
  すでにマッピングされているシグナルについて、シグナルと同名の ASCET メッセージを検索し、見つかった ASCET メッセージにそのシグナルを

マッピングします。同名の ASCET メッセージが見つからない場合、そのメッセージのマッピングは解除され、“Mapping”タブの“ASCET Message”フィールドが空白になります。

- **Create ASCET messages**：
  このオプションをオンにしておくと、足りない ASCET メッセージの代わりとなるグローバルメッセージが生成されます。つまり、上記のいずれのアクションを実行しても、所定のシグナル（全シグナル、またはマッピング済みシグナル）はすべてマッピングされます。

Generate Receive Debug Signals

このオプションをオンにすると、各“receive”シグナルグループごとに 2 つのシグナルがさらに生成されます。

- `<GroupName>_Diag_dT`
- `<GroupName>_Diag_Rec`

...dT シグナルは、前回のメッセージ受信からの経過時間（秒）を示します。

注記

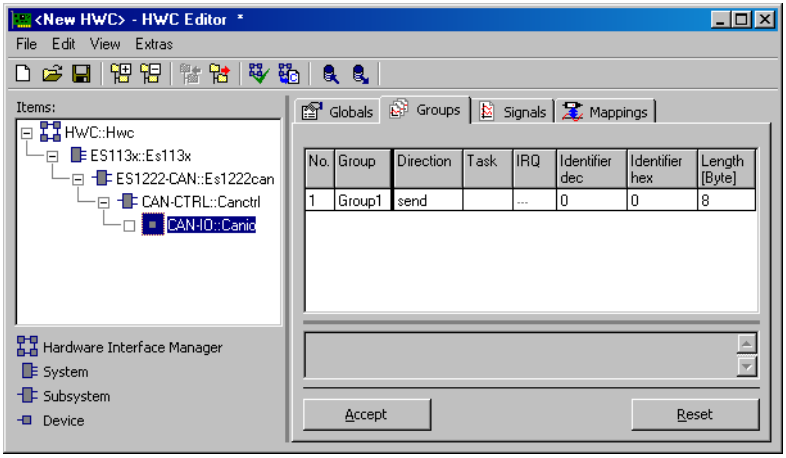
このシグナルを通常のメッセージレシートとして使用（IRQ = `no`）して、受信状態をモニタすることができます。たとえば、CAN バスに割り込みがかかった場合、そのシグナルの値は受信タスクの各周期において増加します。パフォーマンス上の理由から、この値のオーバーフロー（約 300 秒で発生します）についての対策は行われません。

これに対し、メッセージが割り込みにより受信される場合（`IRQ` = `yes`）には、値の計算は割り込みタスクが実行されるまでは行われないので、実際の受信モニタリングを行うことはできません。メッセージが受信されない場合には計算が行われないので、古い値がそのまま保持されます。つまり、割り込みが発生した時に、メッセージの受信が通常のタスク周期で行われたのか割り込みにより行われたのかを判断することはできません。

...Rec シグナルは、メッセージが受信されるたびに `true` になります。このシグナルを送受信メッセージにマッピングして、アプリケーションがこのシグナルが読みとるたびにそれを `false` にリセットするようにすれば、メッセージが各周期の間で受信されたものかどうかをアプリケーションが容易に判断できます。

10.4.4 Groups（CAN-IO デバイス）

このタブ上で、CAN メッセージをシグナルグループとして定義します。



注記

このタブのショートカットメニューを使用して、新しいシグナルグループまたは CAN メッセージを生成することができます（詳細は、105 ページの「“View”メニュー」の項を参照してください）。

注記

グループ名、シグナル名、またはシグナルの方向を変更すると、すでにマッピングされている ASCET メッセージが自動的にはマッピングされなくなる可能性があり、この場合、マニュアル操作でマッピングし直す必要があります。

Direction

CAN メッセージの方向を指定します（“send”＝送信メッセージ、“receive”＝受信メッセージ）。

Task

メッセージの送信または受信を行うタスクを指定します。受信メッセージが割り込みモードで受信される場合には、この設定はリセットされてロックされ、変更できなくなります。

IRQ

受信メッセージを割り込みモードで受信するかどうかを指定します。通常の受信の場合、相手方が送信できる範囲でできるだけ多い回数で、タスク内で CAN メッセージをポーリングする必要があります。この動作モードでは、固定周期で送信

されない CAN メッセージや散発的にしか発生しない CAN メッセージについては問題が発生する可能性があるため、そのような状況では、メッセージが受信されるとメッセージの処理がトリガされる、割り込み受信を行うのが理想的です。

Identifier dec/hex

メッセージ識別子を入力します。入力できる値の大きさは、上位のアイテムである CAN コントローラ（CAN-CTRL）の識別子タイプとして選択された設定（standard または extended）により異なります。

標準識別子：	11 ビット	2 047 dec	7 FF hex
拡張識別子：	29 ビット	536 870 911 dec	F FF FF FF hex

各シグナルグループまたは CAN メッセージの識別子はユニークでなければなりません。

Length [Byte]

転送されるメッセージの有効データバイト数（1 ～ 8）を指定します。

Activated Task

ここに“receive”CAN メッセージ[1]（シグナルグループ）用の software タスクを指定して、該当するシグナルグループが受信されるたびにそれが起動されるようにすることができます。ここで指定されたタスクは、クリーンアップなどのポストプロセッシングを行います。

この項目は、デフォルト状態では非表示になっています。

Prescaler

割り込みモードで受信されるメッセージによるデータ転送を行うための VME バス割り込みをいつトリガするかを設定します。デフォルト設定の“1”の場合、メッセージが受信されるたびにデータ転送も行われます。この値をたとえば“2”にすると、2 回に 1 回のメッセージ受信でしか VME バス割り込みがトリガされなくなり、最後に受信されたメッセージのデータだけが転送されます。

この項目は、デフォルト状態では非表示になっています。

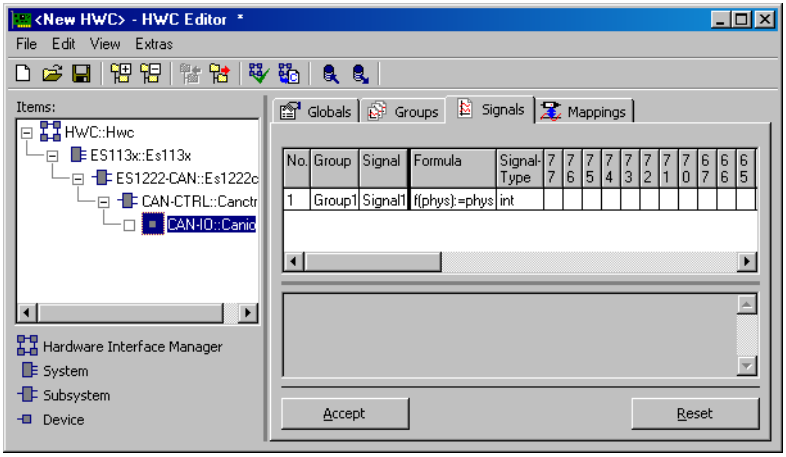
注記

メッセージ受信の頻度が高すぎて VME バス割り込みによる負荷が大きくなりすぎてしまう場合は、この値を大きくしてください。

1. “send" グループ用にもこのようなタスクを指定することは可能ですが、そのような使用法はお勧めできません。（送信プロセスは非同期で行われるため、タスクが起動された時に CAN メッセージの送信が完了しているかどうかが一定でないため。）

10.4.5 Signals（CAN-IO デバイス）

このタブでは、CAN I/O シグナルを定義します。



注記

新しいシグナルは、このタブ用のメニューで生成することができます（詳細は、105 ページの「“View”メニュー」の項を参照してください）。

Group

シグナルを任意のシグナルグループに割り当てます。

Signal Type

シグナルが CAN バスで転送される際のシグナルタイプを指定します。

以下の設定が可能です。

シグナルタイプ	データ型
`int`	符号付きのシグナルをデフォルトの 2 の補数（最大 32 ビット）で表します。
`(s)int`	符号付きシグナルの符号が最上位ビットとして転送されてから、シグナルの絶対値が転送されます。符号ビットが 1 のときには、その数値は負です（最大 32 ビット）。

シグナルタイプ	データ型
uint	符号なしシグナルを表します（最大 32 ビット）。
bool	ブールシグナルを表します。ビットマトリックス内の 1 ビットだけを 1 にセットできます。
real	「標準 IEEE 浮動小数点（4 バイト）」フォーマットの浮動小数点数値を表します。したがって、ビットマトリックス内の最大 32 ビットまでを 1 にセットできます。

IEEE 浮動小数点フォーマットの概要を下の表に示します。

フォーマット	符号	指数部	小数部
Float	1 ビット	8 ビット	24 ビット
Double （サポートされていません）	1 ビット	11 ビット	52 ビット

7654321.. (Bit matrix)

1 つの CAN メッセージで、最大 8 データバイトを転送できます。ビットマトリックスで、各シグナルがどのビットを使用するかを指定します（1 シグナル＝ 1 行）。

列は以下のような構成になります。

7	7	...	0	0	バイト番号
7	6	...	1	0	ビット番号

ビットフィールドの意味：

空のフィールド	シグナルはそのビットを使用しません。
使用されているフィールド	シグナルはこの位置のビットを使用します。
“X”フィールド	シグナルの有効データバイト数が少ないため、このフィールドに対応するビットはデータ転送には使用できません（“Groups”タブの“Length”を参照してください）。

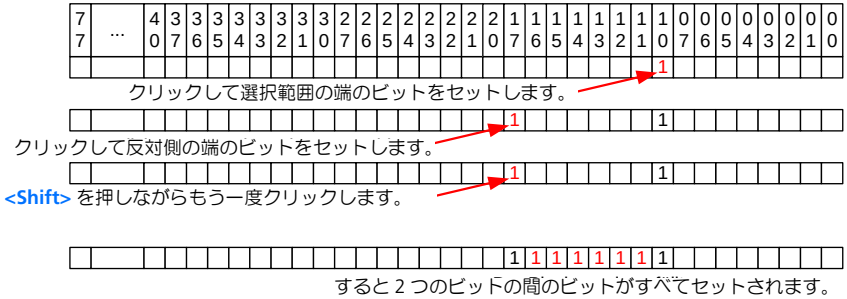
ビットフィールドの操作：

キーボードの矢印キーを使用して、操作するビットセルを選択します。

数字キーで割り当てる値を入力します。この値は、以降に説明されている「ブロック指定」の際に重要となります。

マウス操作の場合、マウスをクリックするとセルの“empty”と“1”が切り替わり、**<Shift>** キーを押しながらマウスをクリックすると値が“1”から“9”まで 1 ずつ順に増加します。

一度に複数のビットを選択するには、次の図のように行います。



セルに異なる値（“1111　2222...”）を入力することによってデータブロックを作成することができ、これにより、事実上あらゆるフォーマットの転送シグナルを指定できます。データブロックを構成するために使用される数値には意味があり、転送時には、最大値（“2222”）を持つブロックには上位ビットが含まれ、最小値（“1111”）を持つブロックには下位ビットが含まれます。1 ～ 9 の範囲の数字を使用して、最大 9 個のビットブロックで 1 つのシグナルを作成することができます。ビット配置は Intel のシグナル転送形式と同じなので、Motorola フォーマットで 8 ビットより長いシグナルを表現するためには、必ずブロック指定を行う必要があります。

さまざまなシグナル定義の例：

10.4.6　Mappings（CAN-IO デバイス）

このタブ上の設定は、すべてのデバイスについて共通です。125 ページの 7.4.5 を参照してください。

10.5 ES1222-CAN Bypass（CAN バイパスプロトコル - CBP）

CAN バイパスアプリケーションは、ETK バイパスアプリケーションほど効率的ではありませんが、その反面、ECU ソフトウェアを CAN バイパスに対応させるだけで実現でき、特殊なハードウェアを必要としません。

ただし、ECU に CAN インターフェースが装備されていることが条件となります。CAN バイパスを行うためには以下の 2 つの方法があります。

- ECU ソフトウェアで固定的に定義されたメッセージ、および割り当てられたシグナルによる方法。これは CAN バイパスデバイスではなく CAN-IO デバイスを使用します（166 ページの 10.4 項を参照してください）。
- 可変なメッセージサイズを転送できるプロトコルによる方法。この場合、次の項のライセンス合意条件に留意してください。

10.5.1 CAN バイパスプロトコル（CBP）のライセンスに関する法的な注意事項

CAN バイパスプロトコル（CBP）は、Robert Bosch GmbH Stuttgart によって開発され、Robert Bosch GmbH および ETAS GmbH の両者によって応用されています。すべての権利は Robert Bosch GmbH に留保されています。

CAN バイパスは、このプロトコルが組み込まれた ASCET-RP および Robert Bosch GmbH 製 ECU と共に使用する場合に限り、更なるライセンス合意なしに使用することができます。

CBP をサポートする ECU についての情報は、ECU メーカーから入手できます。

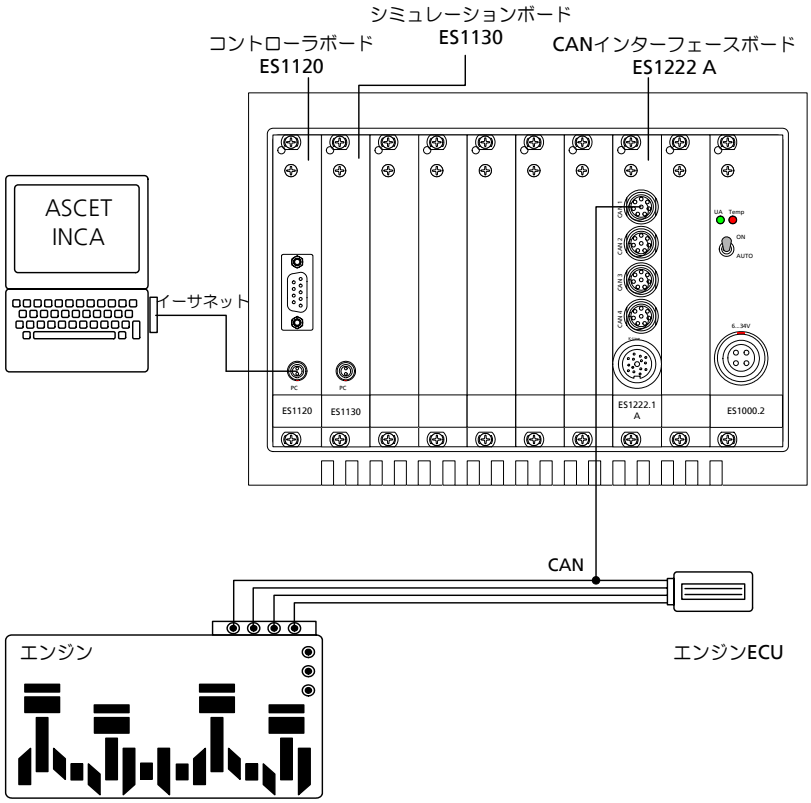
ASCET-RP と共に使用する CBP のインターフェースディスクリプションについては、ご要望に応じて ETAS GmbH からお渡しできます。

注記

インターフェースディスクリプションの譲渡は、定められた譲渡合意のもとに行われます。ASCET-RP のライセンス合意からこの譲渡合意への権利授与はありません。

10.5.2 CAN バイパスのハードウェア構成

下図に、ES1000.2 ／ ES1000.3 を用いた CAN バイパスアプリケーションの構成例を示します。



ETK による高速プロトタイピングアプリケーションの場合と同様に、ホスト PC はホストリンクインターフェース経由で ETAS 実験システムに接続されます。ASCET で開発されたバイパスファンクションは PPC モジュール（ES113x）上で実行され、ECU は、ES1222 CAN インターフェースボード経由で ETAS 実験システム（ES1000.x）に接続されます。

ECU との通信は、CAN バイパスプロトコル（CBP）を使用して行われます（CBP の著作権は Robert Bosch GmbH に留保されています）。

10.5.3 Globals（CAN-Bypass デバイス）

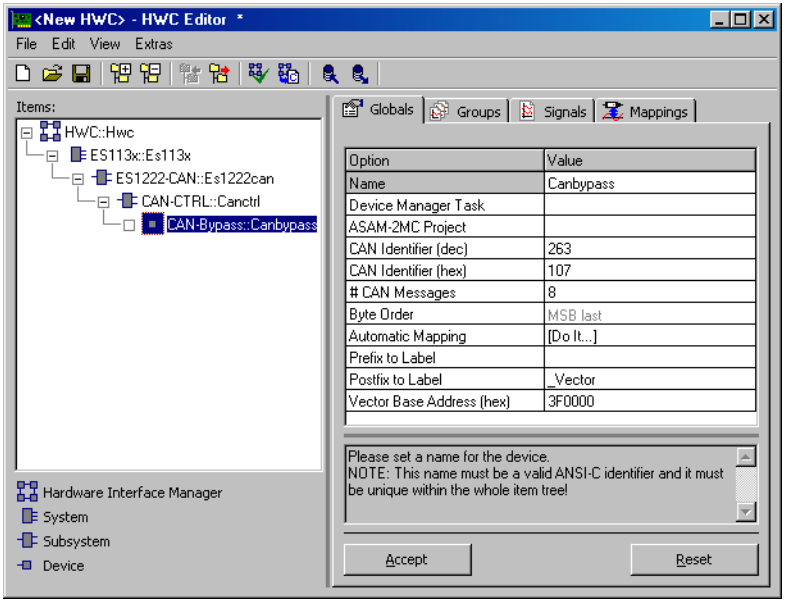


図 10-19 CAN Bypass デバイスの“Globals”タブ

Device Manager Task

バイパスシステムで排他的に使用される Alarm タスクを指定します。この Alarm タスクの時間周期は、0.05 ～ 0.8 秒にしてください。

ASAM-2MC Project

すでにデータベースに読み込まれている、ECU に適した ASAM-MCD-2MC プロジェクトを選択します。

CAN Identifier (dec)

CAN メッセージコマンドの CAN 識別子を 10 進表記で指定します。CAN メッセージコマンドは、CAN メッセージブロックの最大値の識別子を持つメッセージです（CBP のデフォルト値は 263 です）。識別子の長さ（11 ビット／ 29 ビット）は、CAN-CTRL サブシステムの“Globals”タブ上の“Identifier”の設定により決まります。

CAN メッセージブロックのフォーマットとコマンドメッセージは、CBP インターフェースディスクリプションに定義されています。

CAN Identifier (hex)

CAN メッセージコマンドの CAN 識別子を 16 進数で指定します（CBP のデフォルト値は 107H です）。

 # CAN Messages

CAN メッセージブロックに含める CAN メッセージの数（4 ～ 16）を指定します。CAN バイパスプロトコルでは、通常、CAN メッセージブロックには 8 つの CAN メッセージが含まれます。

注記

CAN メッセージの数が多ければ多いほど、たくさんのバイパス変数を転送できます。ただし、ECU ソフトウェア側でもより多くのリソースが必要となるため、この項目を設定する際は、ECU ソフトウェアの調整が必要です。

Byte Order

ECU プロセッサのワードデータのストレージフォーマット（MSB first/MSB last）が表示されます。この情報は、割り当てられている ASAM-MCD-2MC プロジェクトから読みとられたものです。

“Byte Order”の一般的な設定：

MSB first	big endian	モトローラ
MSB last	little endian	インテル

Automatic Mapping

173 ページの「Automatic Mapping」を参照してください。

Prefix / Postfix to Label

バイパスパラメータをバイパス出力値（ASAM-MCD-2MC の Measurement）と区別できるようにするための、名前の拡張子を指定します。この名前の形式によってバイパス出力値（ECU への“send”シグナル）であることが認識され、バイパスシグナル用に必要なすべての情報が決定されます。

この関係を下図に示します。

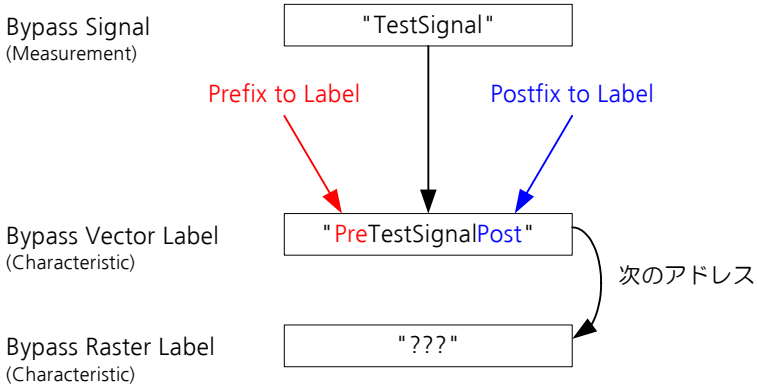


図 10-20 バイパスラベル間の関係

バイパスラベルの役割：

Bypass Vector Label	バイパス変数のベクタアドレスを指定します。0 の場合、値は転送されません。
Bypass Raster Label	バイパス変数を角度同期ラスタで転送する（値＝ 0）か、時間同期ラスタで転送する（値＝ 1）かを指定します。

Vector Base Address (hex)

ECU のバイパス出力値（ECU の“send”シグナル）のターゲットアドレスの計算に使用されるベースアドレスを指定します。これは、ターゲットアドレスが CAN バイパスプロトコルで使用可能な 16 ビットでアドレシングできる領域の外にある場合に必要です。

ターゲットアドレスは、以下のようにして計算されます。

```
<bypass address> :=
      <vector address> + <ecu calibration offset> -
                                          <base address>
```

`<ecu calibration offset>` は、ASAM-MCD-2MC プロジェクトにより決められます。

10.5.4 Groups（CAN-Bypass デバイス）

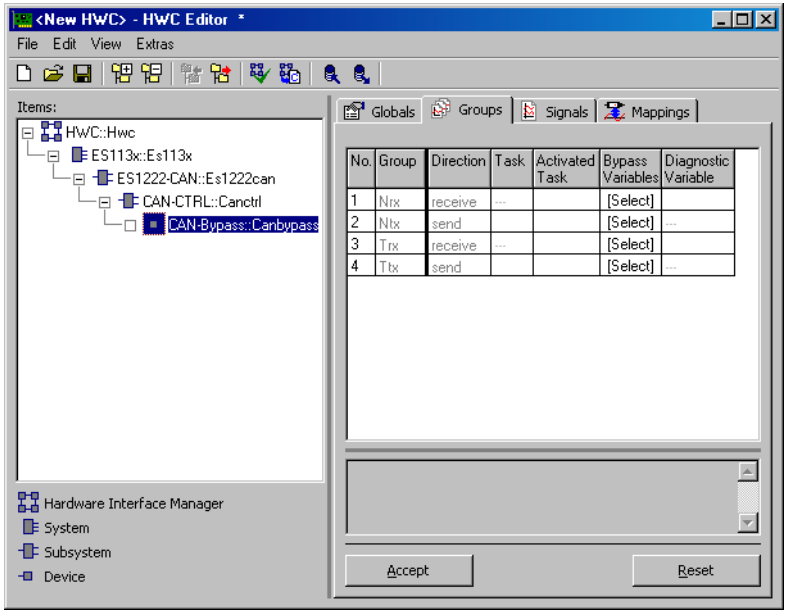


図 10-21 CAN-Bypass デバイスの“Groups”タブ

注記

グループ名、シグナル名、またはシグナルの方向を変更すると、すでにマッピングされている ASCET メッセージが自動的にはマッピングされなくなる可能性があり、この場合は、マニュアル操作でマッピングし直す必要があります。

Group

CAN-Bypass デバイスは、send 方向と receive 方向の最大 2 つのラスタをサポートします。シグナルグループには以下の意味があります。

Nrx　ECU から CAN-Bypass デバイスへの速度同期データ

Ntx　CAN-Bypass デバイスから ECU への速度同期データ

Trx　ECU から CAN-Bypass デバイスへの時間同期データ

Ttx　CAN-Bypass デバイスから ECU への時間同期データ

Activated Task

速度同期または時間同期のデータが受信されたときに起動されるソフトウェアタスクを指定します。バイパスファンクションの演算および ECU への結果の返送も、これらのタスクで行われます（“Task”列の対応する項目を参照してください）。

Bypass Variables

この項目のセルを選択すると、Select Variables ダイアログボックスが開き、シグナルグループに割り当てるバイパス変数（Measurement）を選択することができます。

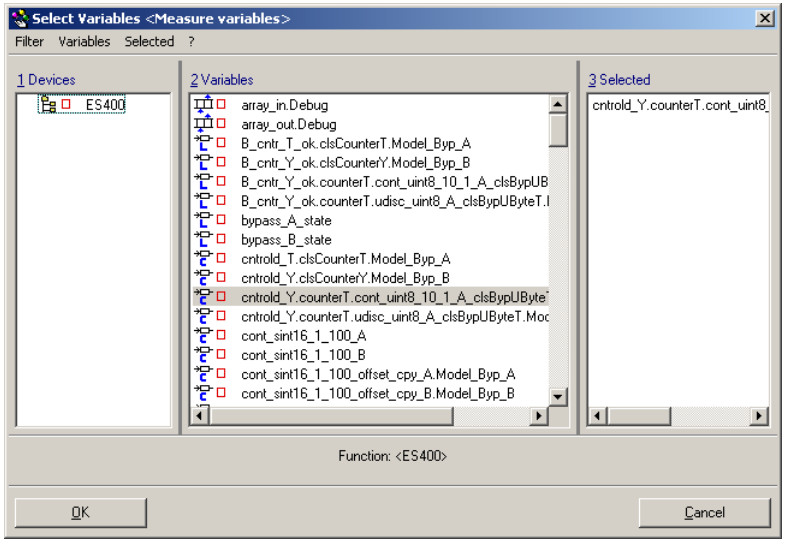


図 10-22 “Select Variables”ダイアログボックス

注記

ノンリニアな変換式はサポートされていません。そのような変換式が選択されると、以下のようなワーニングメッセージが表示されます。

<RTIO Toolbox WARNING >
Error on processing signal ’<signal_name>’ formula: Can’t handle ’<formula_name>’ for bypass; formula is converted to ’FormulaId’!

基本的に、ASAM-MCD-2MC プロジェクトで定義されている Measurement（測定変数）はすべて“receive”シグナルグループに割り当てることができます。また、ECU ソフトウェアによりバイパス変数としてサポートされていて、その名前

の形式（“Globals”タブの“Prefix/Postfix to Label”を参照してください）から明らかにバイパス変数であることがわかる測定値に限り、“send”シグナルグループに使用できます。

注記

使用できる測定値の数は限られていて、測定値のデータ型（byte、word...）や使用できるバイパス CAN メッセージの数（“Globals”タブの“# Bypass Messages”という項目）により異なります。

Diagnostic Variable

必要に応じて、各“receive”シグナルグループに ECU 内のその他の値を割り当て、診断用（例、バイパス動作をモニタする）に利用することができます。

“Bypass Variables”の項と同じ方法で選択します。

10.5.5 Signals（CAN-Bypass デバイス）

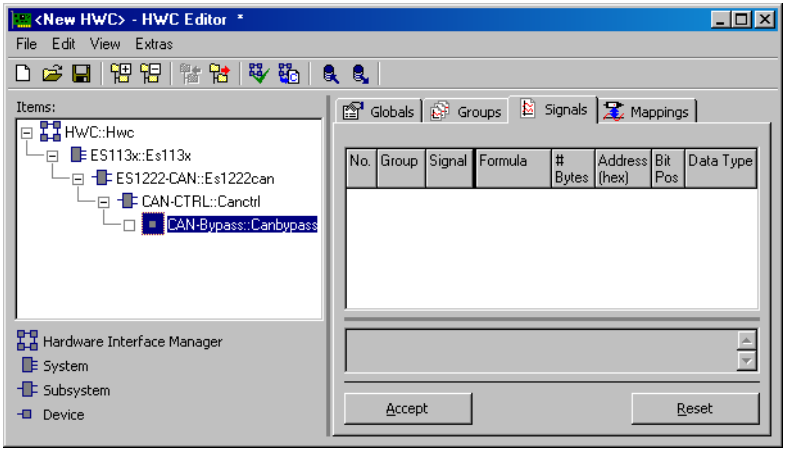


図 10-23 CAN-Bypass デバイスの“Signals”タブ

注記

“Signals”タブの項目にはバイパス変数のステータス値が表示され、編集はできません。

バイパス変数は、シグナルグループ内でバイトサイズに応じてソートされます（8-4-2-1 バイト）。

 # Bytes

バイパス変数 1 つ当たりのバイト数が表示されます。

Address (hex)

バイパス変数のアドレスが表示されます。

receive 値のアドレススペースは 32 ビットで、send 値のアドレススペースは 16 ビットです。

Bit Pos

現在のところ、標準の ECU ソフトウェアはビット値のバイパスをサポートしていないので、この項目は未使用です。

Data Type

バイパス変数のデータ型が表示されます。

Byte Offset

バイパス変数のシグナルグループ内のオフセットが表示されます。

10.5.6 Mappings（CAN-Bypass デバイス）

このタブ上の設定は、すべてのデバイスについて共通です。125 ページの 7.4.5 項を参照してください。

10.6 ES1223-LIN

ES1223 ボードは、VME バスシステムの LIN インターフェースとして使用されます。このボードは 4 つの LIN チャンネルを搭載しています。受信したメッセージで VME バス割り込みをトリガすることが可能なので、メッセージのポーリングを行なわないようにすることができます。各 LIN コントローラは、それぞれ最大 64 のメッセージを処理できます。専用のプロセッサが使用されるので、システムコントローラの負荷は大幅に軽減されます。

本項では、ES1223 ボードの RTIO への組み込みについて説明します。HWC エディタに ES1223 ボードを組み込むには、“ES1223-LIN”というアイテムを選択します。

10.6.1 Globals（ES1223-LIN サブシステム）

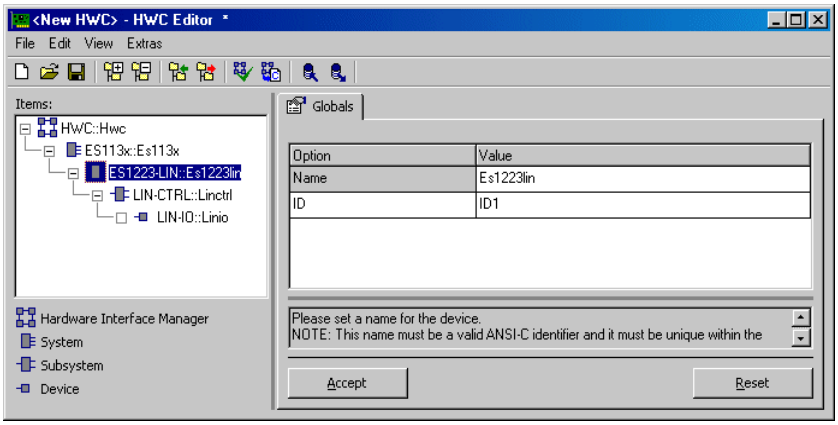


図 10-24 ES1223-LIN サブシステムの“Globals”タブ

ID

使用するボートのボート番号を入力します。

注記

ES1223 は、システムがオンになったときにハードウェアマネージャにより ID 番号およびフリーアドレス領域が自動的に割り当てられる、“Auto-ID”ボードと呼ばれるタイプのボードです。同じタイプのボードが複数存在する場合には、左から右の順に番号が付けられます。つまり、一番左のボードの番号が 1 になり、その次の番号は 2 になります。1 つの実験システムには最大 2 枚のボードを使用することができ、2 種類の ID を指定できます。

10.6.2 Globals（LIN-CTRL サブシステム）

LIN-CTRL サブシステムの“Globals”タブに、”Init Task”、“Exit Task”、“IRQ Handler Task”を割り当てます。

ボード上の 4 チャンネルの LIN インターフェースを利用し、1 つの LIN-CTRL サブシステムに最大で 4 つの LIN-IO デバイスを割り当てることができます。

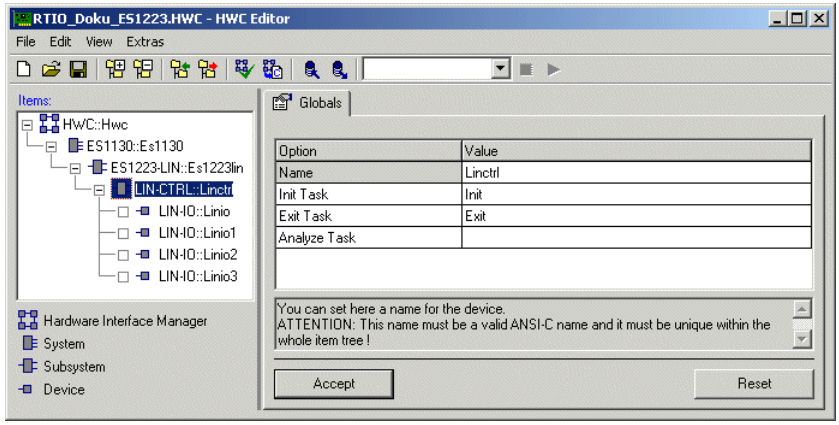


図 10-25 LIN-CTRL サブシステムの“Globals”タブ

IRQ Handler Task

“IRQ Handler Task”は、ES1223 ボードの割り込みを使用する場合に必要です。このタスクは、ASCET プロジェクトエディタのタスクリスト内に、“Software”タスクとして定義されている必要があります。“Max. No. of Activations”フィールドには 2 以上の数（最大 50）を入力してください。

10.6.3 Globals（LIN-IO デバイス）

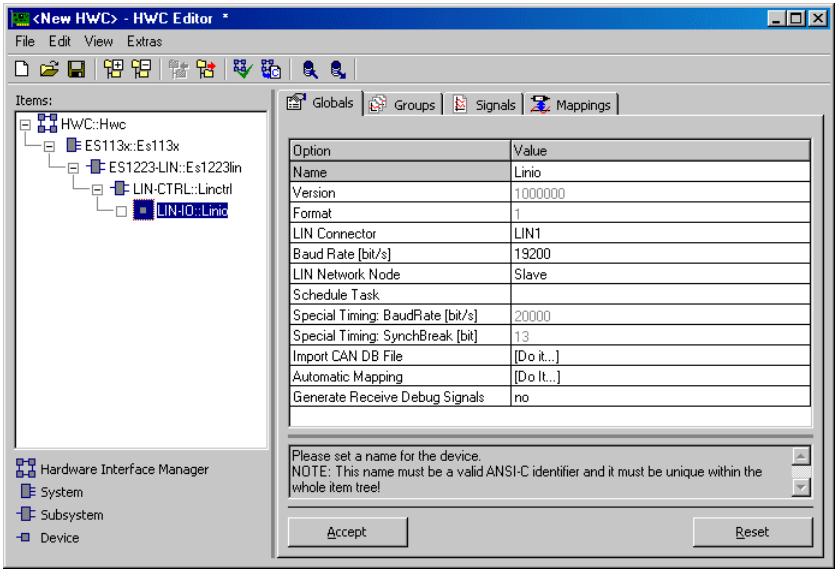


図 10-26 LIN-CTRL サブシステムの“Globals”タブ

LIN Connector

使用する LIN コネクタ（LIN1 ～ LIN4）を選択します。各 LIN-IO デバイスをそれぞれのコネクタに接続してください。

Baud Rate [bit/s]

転送レートを指定します。標準的なボーレート（19200、9600、2400 bit/s）から選択できます。また、**<Special Timing>** 設定を選択し、LIN コントローラのビットタイミングおよび転送レートに関する低水準の制御をアクティブにすることもできます。

“Special Timing”設定を指定するためには、192 ページの「Special Timing」についての記述を参照してください。

LIN Network Node

選択された LIN-IO デバイスをマスタユニットまたはスレーブユニットのどちらで稼動させるかを指定します。1 つの LIN ネットワーク内に必ず 1 つのマスタユニットが必要です。

Schedule Task

この設定は、“LIN Network Node”が `Master` に設定されている LIN-IO デバイスにのみ意味を持ちます。送信キューが空である時に常に実行されるマスタコントロールユニット用タスクを定義するもので、メッセージの連続送信を行う場合に有効です。

Special Timing: BaudRate [bit/s]

このオプションはデフォルトでは非表示になっています。“Baud Rate”オプションの値が `<Special Timing>` になっている場合に限り、編集できます。

標準の転送レート（191 ページの「Baud Rate [bit/s]」を参照してください）とは異なり、10 ～ 20,000 bit/s の値を任意に設定できます。

Special Timing: SynchBreak [bit]

このオプションはデフォルトでは非表示になっています。“Baud Rate”オプションの値が `<Special Timing>` になっている場合に限り、編集できます。

ここで設定される SyncBreak 長には、実際の SyncBreak（Low フェーズ）と同期デリミタ（1 ビット）が含まれます。標準の設定（最低 14 ビット）以外に、8 ～ 15 ビット（低速）の間で任意に同期長を設定できます。

Import CAN DB File

CANdb データ管理プログラムを使用して作成された CAN/LIN データベースファイルをインポートします。このファイルを使用して、LIN メッセージやシグナルを自動生成することができます。

インポートの詳しい方法については、170 ページの「Import CAN DB File」を参照してください。

Automatic Mapping

シグナルと ASCET メッセージのマッピングを自動的に行います。

詳しい情報は、173 ページの「Automatic Mapping」を参照してください。

Generate Receive Debug Signals

このオプションがアクティブ（= `yes`）になっていると、各“receive”シグナルグループに対して以下の 2 つの補助シグナルが生成されます。

- `<GroupName>_Diag_dT`
- `<GroupName>_Diag_Rec`

詳しい情報は、174 ページの「Generate Receive Debug Signals」を参照してください。

10.6.4 Groups（LIN-IO デバイス）

シグナルグループごとに LIN メッセージを定義します。

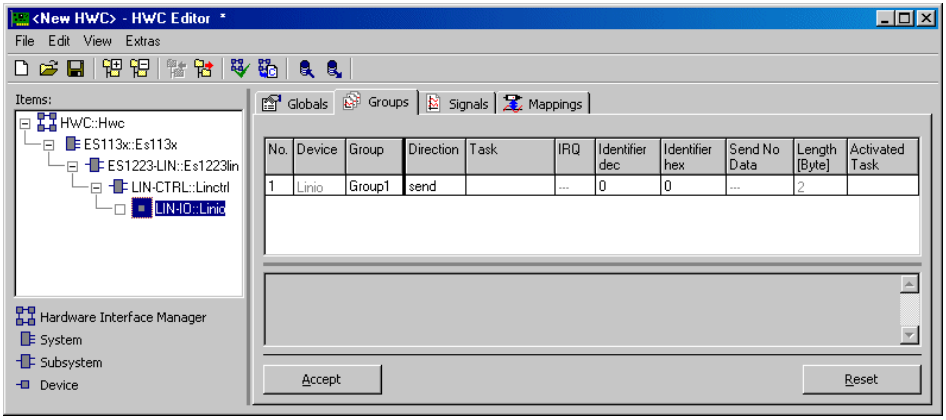


図 10-27 LIN-IO デバイスの“Groups”タブ

“Groups”タブ内のショートカットメニューを使用して、新しいシグナルグループ、または LIN メッセージを作成できます（105 ページの「“View”メニュー」の項を参照してください）。

注記

グループ名、シグナル名、またはシグナルの方向を変更すると、すでにマッピングされている ASCET メッセージが自動的にはマッピングされなくなる可能性があり、この場合、マニュアル操作でマッピングし直す必要があります。

各オプションの詳しい内容については、10.4.4「Groups（CAN-IO デバイス）」を参照してください。LIN-IO デバイスには以下のような特徴があります。

- LIN-IO デバイスの場合、LIN メッセージ長は識別子の中にコード化されて含まれているため、“Length [Byte]”オプションはデフォルト状態ではマスクされていて、書き込み不可能です。
- “Prescaler”オプションは使用できません。
- “Send No Data”オプションは LIN 固有のものなので、以下に説明します。

Send No Data

このオプションは、“Globals”タブの“LIN Network Node”オプションが `Master` に設定されていて、かつ“Groups”タブの“Direction”の項目で `send` が選択されている場合のみ使用されます。

“Send No Data”オプションを Yes に設定すると、メッセージヘッダのみが転送され、データは転送されません。この場合、メッセージ長は、Length [Byte] = 0 となります。

注記

マスタノードでスレーブノードからのデータを受信するには、**同じ識別子**を持つ 2 つのシグナルグループを作成する必要があります。

- 方向が“send”で“Send No Data”が設定されたグループ
- 方向が“receive”であるグループ

マスタノードは最初のシグナルグループでメッセージヘッダを送信し、次にスレーブノードからのデータを受信します。

10.6.5 Signals（LIN-IO デバイス）

このタブで、LIN シグナルを定義します。

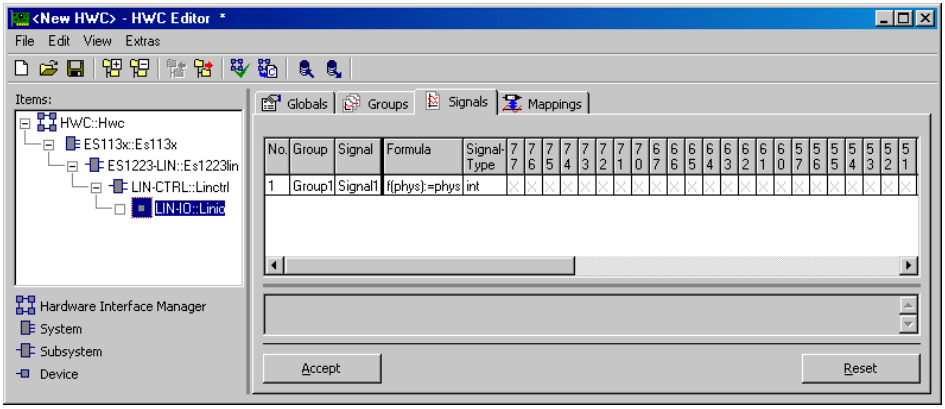


図 10-28 LIN-IO デバイスの“Signals”タブ

“Signal”タブ上のショートカットメニューで、新しいシグナルを作成できます（105 ページの「“View”メニュー」を参照してください）。

テーブルの各項目の内容については、177 ページの 10.4.5「Signals（CAN-IO デバイス）」を参照してください。

注記

メッセージ長は、“Group”タブで設定する識別子によって自動的に決まります。識別子の上位 3 ビットによりメッセージ長（2、4、8 バイト）が決まるので、ビットフィールドにおいてこれらの上位ビットは無効です。

10.6.6 Mappings（LIN-IO デバイス）

このタブ上の設定は、すべてのデバイスについて共通です。125 ページの 7.4.5 を参照してください。

10.6.7 ランタイムのオプション

LIN バスは ES1223 プロセッサによって割り込みモードで使用されます。この割り込み管理のため、送受信コマンドに 250 µs までのディレイを持たせることができます。

10.7 ES1231-ETK

ETK インターフェースボードの ES1231 は ES1200 ボードの後継モデルで、ES1200 と同様、ETK 搭載の ECU を実験システムにリンクさせるために使用されます。

このボードでは、ブロック転送モードの導入や転送レートの向上など、先代モデルに比べて機能が強化されていて、はるかに高速なデータ転送を行なうことができます。このボードは、メモリを節約できる“Auto-ID”技術も装備しています。

注記

ES1231 ボードには、ETK と通信するための特別なシステムサービスが必要です。このサービスは ES1120（VCU2）システムコントローラボードによってのみ提供されます（67 ページの「ハードウェア - ES1000.x 実験システム」という項を参照してください）。

10.7.1 Globals（ES1231-ETK サブシステム）

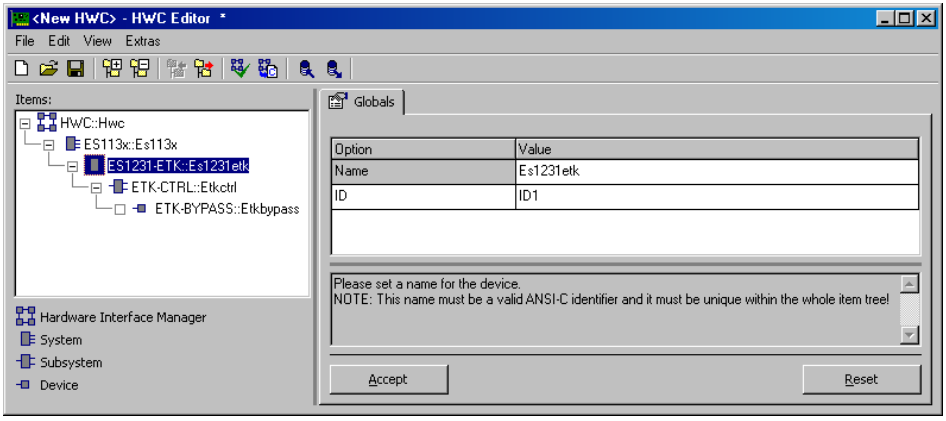


図 10-29 ES1231-ETK アイテムの“Globals”タブ

ID

ボード番号を入力します。

注記

ES1231 ボードは、システムがオンになったときにハードウェアマネージャにより ID 番号およびフリーアドレススペースが自動的に割り当てられる“Auto-ID”ボードと呼ばれるタイプのボートです。同じタイプのボードが複数（例、ES1231 が 2 枚）ある場合には、左から右の順に番号が付けられます。つまり、一番左のボードの番号が 1 になり、次の番号が 2 になります。1 つの実験システムでは最大 4 枚のボードを稼働させることができるため、4 種類の ID を指定できます。

ES1231 には ETK コントローラが 1 基しかないので、“ETK-CTRL”アイテムは 1 つしか挿入できません。使用できるポートはポート A のみです。

10.7.2 Globals（ETK-CTRL サブシステム）

151 ページの「Globals（ETK-CTRL サブシステム）」という項を参照してください。

10.7.3 Globals（ETK-BYPASS デバイス）

153 ページの「Globals（ETK-BYPASS デバイス）」という項を参照してください。

10.7.4 Groups（ETK-BYPASS デバイス）

160 ページの「Groups（ETK-BYPASS デバイス）」という項を参照してください。

10.7.5 Signals（ETK-BYPASS デバイス）

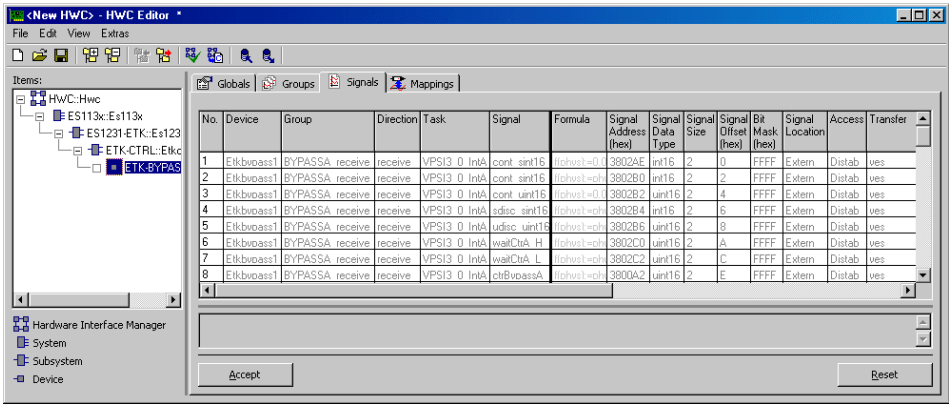


図 10-30 ETK-BYPASS デバイスの“Signals”タブ

注記

“Signals”タブにはバイパス変数用ステータス値が表示されます。このタブにはユーザーが編集できる項目はありません。

この“Signals”タブには、10.3.5「Signals（ETK-BYPASS デバイス）」の項で説明されている ETK バイパスの“Signals”タブの内容に加え、さらに以下の 2 つのカラムが含まれています。

Bit Mask (hex)

ES1231 は、ETK コントローラから 1 バイト（8 ビット）または 1 ワード（16 ビット）ずつデータを読み込みますが、ASAM-MCD-2MC および HWC エディタでは、それ以外にビットシグナル（1 ビット）を使用することができます。この“Bit Mask (hex)”フィールドに入力されたビットマスクと ECU から読み込んだ値が AND 演算されてビットシグナルとなります。

1 バイトの値からは、8 種類のマスク（`0x1`、`0x2`、`0x4`、`0x8`、`0x10`、`0x20`、`0x40`、`x80`）によって 8 個のビットシグナルが作成されます。また FAR アドレスの場合、マスク（`0x00FF` または `0xFF00`）によって、1 つのワード値から 2 つのバイト値が作成されます。

このカラムはデフォルトでは非表示になっています。

Transfer

1 つのバイト値から複数のシグナルが生成される際、システムはその値がすでに読み込まれている値であることを認識すると、転送時間を節約するため、その値のローカルコピーを使用します。この機能を使用するには、1 つのアドレス内の先頭のシグナルのみ“Transfer”カラムを Yes にし、他のシグナルは No にします。

10.7.6 Mappings（ETK-BYPASS デバイス）

このタブ上の設定は、すべてのデバイスについて共通です。125 ページの 7.4.5 項を参照してください。

10.8 ES1232-ETK

ETK インターフェースボードの ES1232 は ES1231 の後継モデルで、実験システムに 1 台の ECU と ETK をリンクするために使用されます。

このボードは、ES1231 の機能がさらに拡張されていて、間接アドレッシングの機能を持ち、転送レートは 8Mbit または 100Mbit を使用できます。また最大 32 の測定ラスタに対応し、新しいバージョンの ASAM-MCD-2MC ファイル（AML V1.2.0 ）にも対応しています（10.8.5「Globals（ETK-CTRL-ADV サブシステム）」の項を参照してください）。また、シリアル ETK も使用できます。

10.8.1 Globals（ES1232-ETK サブシステム）

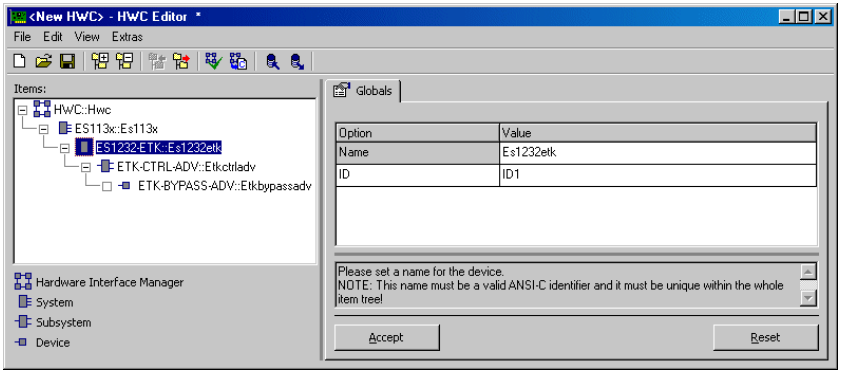


図 10-31 ES1232-ETK アイテムの“Globals”タブ

ID

ボード番号を入力します。この番号は VME システム内のボードの並び順を表わします。たとえば 1 番左側に挿入された ES1232 には **ID1** を選択し、2 番目の ES1232 には **ID2** を選択してください。

注記

ES1232 ボードは、システムがオンになったときにハードウェアマネージャにより ID 番号およびフリーアドレススペースが自動的に割り当てられる“Auto-ID”ボードと呼ばれるタイプのボードです。同じタイプのボードが複数（例、ES1232 が 2 枚）ある場合には、左から右の順に番号が付けられます。つまり、一番左のボードの番号が 1 になり、次の番号が 2 になります。1 つの実験システムでは最大 4 枚のボードを稼働させることができるため、4 種類の ID を指定できます。

ES1232 には ETK コントローラが 1 基しかないので、この下のレベルには“ETK-CTRL”アイテムを 1 つしか挿入できません。使用できるポートはポート A のみです。

10.8.2 ETK-CTRL-BAS サブシステム

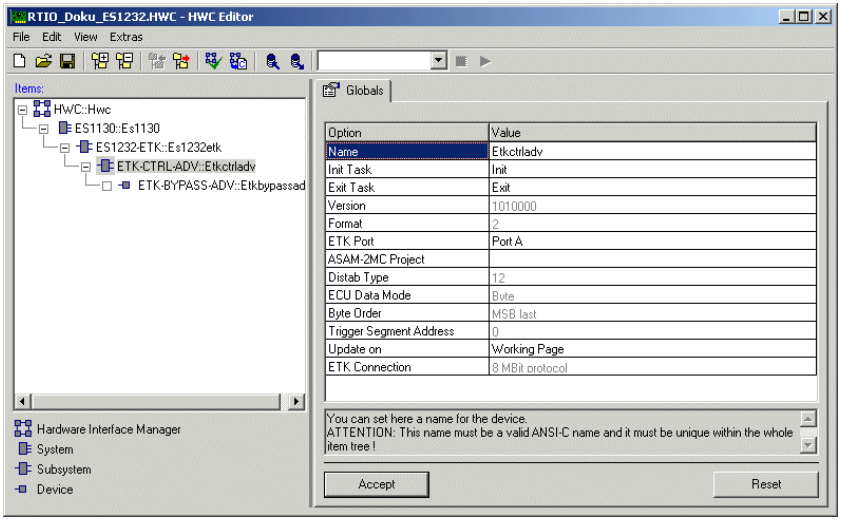


図 10-32 ETK-CTRL-BAS サブシステムの“Globals”タブ

ETK-CTRL-BAS の設定方法は、ES1201 ETK デバイスの ETK-CTRL アイテム（195 ページ、10.7「ES1231-ETK」の項を参照してください）の場合と同じですが、さらに以下の“ETK Connection”オプションが含まれます。

ETK Connection

ボードの転送レート（8 Mbit/s または 100 Mbit/s）が表示されます。この値は ASAM-MCD-2MC ファイルから読み込まれたもので、ここで変更することはできません。

10.8.3 ETK-BYPASS デバイス

ETK-CTRL-BAS サブシステムの ETK-BYPASS デバイスの設定方法は、ES1231 ETK ボードの ETK-BYPASS アイテムと同じです。10.7「ES1231-ETK」の項を参照してください。

10.8.4 旧バージョンのプロジェクトを 100 Mbit/s で使用する（ETK-CTRL-BAS サブシステム）

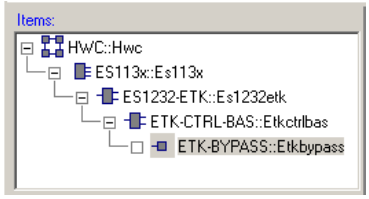
ES1231 で使用していた旧バージョン（AML V1.1.0）のプロジェクトを、ES1232 および ETK-CTRL-BAS サブシステム環境で使用することができます。

転送レートは ASAM-MCD-2MC ファイル（`*.a2l`）の `TP_BLOB` で定義され、AML V1.1.1 以降のバージョンでは、`INTERFACE_SPEED` パラメータが使用されます。

旧バージョンのプロジェクトを使用する場合、高速レート（100 Mbit/s）を使用しなければ、ASAM-MCD-2MC ファイルを変更する必要はありません。以下のように設定してください。

旧バージョンのプロジェクトを 8 Mbit/s で使用する：

- HWC エディタで、以下のようなアイテムツリーを作成します（7.2 項と 11.2.3 項を参照してください）。

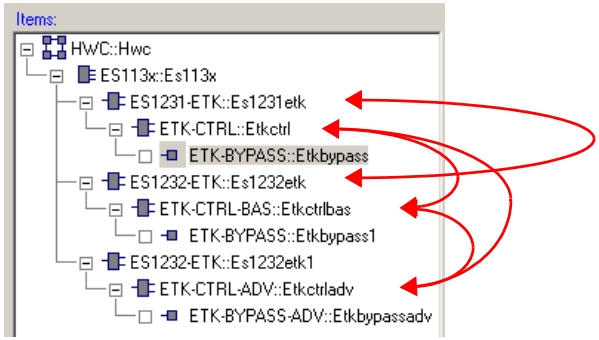


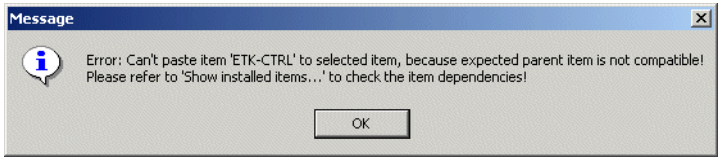
- **Edit → Copy → Item(s)** および **Hardware → Paste → to selected Item** を使用して、今まで使用していた ES1231 の ETK-BYPASS デバイスを、そのまま ES1232 にコピーします。

- **Edit → Copy → Item Data** および**Edit→Paste → Data to selected Item** を使用して、ES1231 の ETK-BYPASS のデータを ES1232 にコピーします。

注記

ES1231 と **ETK-CTRL-BAS サブシステムを組み込んだ ES1232** の間でのみ、**ETK-BYPASS デバイス**または**そのデータ**のみをコピーすることができます。他のアイテムやデータをコピーしようとすると、エラーメッセージが表示されてコピーは行なわれません（次の図を参照してください）。





- ETK-BYPASS デバイス（ES1232/ETK-CTRL-BAS）の“Globals”タブの“ASAM-2MC Project”フィールドをクリックして、ASAM-MCD-2MC ファイルを選択します（**154 ページの「ASAM-2MC Project」**を参照してください）

  ASCET モニタウィンドウに、以下のようなワーニングメッセージが表示されます。

  ```
  Parameter 'Interface Speed' is missing in A2L/TP_BLOB: old TP_BLOB version 0x1000001. Using default: 8 MBit
  ```

  ETK-CTRL-BAS アイテムの“Globals" タブ上の“ETK Connection”の行には、転送レート 8 Mbit/s が表示されます。

ES1231 を使用していた旧バージョンのプロジェクトに高速レート（100 Mbit/s）を適用するには、ASAM-MCD-2MC ファイルの内容を変更し、HWC エディタでも設定を変更する必要があります。

旧バージョンのプロジェクトを 100 Mbit/s で使用する：

- HWC エディタで、必要なアイテムツリーを作成します（200 ページを参照してください）。
- テキストエディタで ASAM-MCD-2MC ファイルを開きます。

  AML V1.1.0 では、TP_BLOB は以下のように定義されています。

```
/begin TP_BLOB 0x1000001 0x0 0x0
   /begin DISTAB_CFG  0xC 0x1 MSB_LAST 0x383000 0x0
      TRG_MOD 0xB7
   /end DISTAB_CFG
   CODE_CHK
   0x0
   0x0
   0x0
   0x0
   ETK_CFG 0xF 0xF0 0xFF 0x3 0xFD 0xEE 0xFF 0x1
   RESERVED 0x810000 0x8103F9 EXTERN 0x-1 0x-1 0x-1
            0x-1 0x-1
/end TP_BLOB
```

この TP_BLOB の第 1 行目は、わかりやすいようにコメントを加えると、以下のようになります。

```
/begin TP_BLOB
     0x1000001     /* TP_BLOP version         */
     0x0           /* Project Base Address    */
     0x0           /* RstCfg parameter (MPC) */
```

- 100 Mbit/s で転送を行なうには、TP_BLOB のバージョンを新しくする必要があります。そのためには、Project base Address パラメータを INTERFACE_SPEED パラメータに置き換えます。

注記

100 MBit/s を使用するための上記の変更は、必ず **IF_DATA ETK** セクションの TP_BLOB について行なってください。

AML V1.1.1/V1.2 では、この行を以下のように変更してください。

```
/begin TP_BLOB
   0x1000100  /* TP_BLOP version */
   2          /* InterfaceSpeed 1=8MBit, 2=100MBit */
   0x0        /* RstCfg parameter (MPC) */
```

- ファイルを保存します。
- コンポーネントマネージャのメニューで **Project → Read ASAM-2MC** を選択して、今変更したファイルをデータベースに読み込みます。
- HWC エディタで、ETK-BYPASS デバイス（ES1232/ETK-CTRL-BAS）の“Globals”タブを開きます。
- “ASAM-2MC Project”のフィールドをクリックして、今インポートした ASAM-MCD-2MC プロジェクトを選択します（**154 ページの「ASAM-2MC Project」**を参照してください）。

  ETK-CTRL-BAS アイテムの“Globals" タブ上の“ETK Connection”の行には、転送レート 100 Mbit/s が表示されます。

10.8.5 Globals（ETK-CTRL-ADV サブシステム）

ETK-CTRL-ADV サブシステムの“Globals”タブで、物理的な ETK コントローラ（ETK ポート）を ETK-CTRL サブシステムに割り当てます。

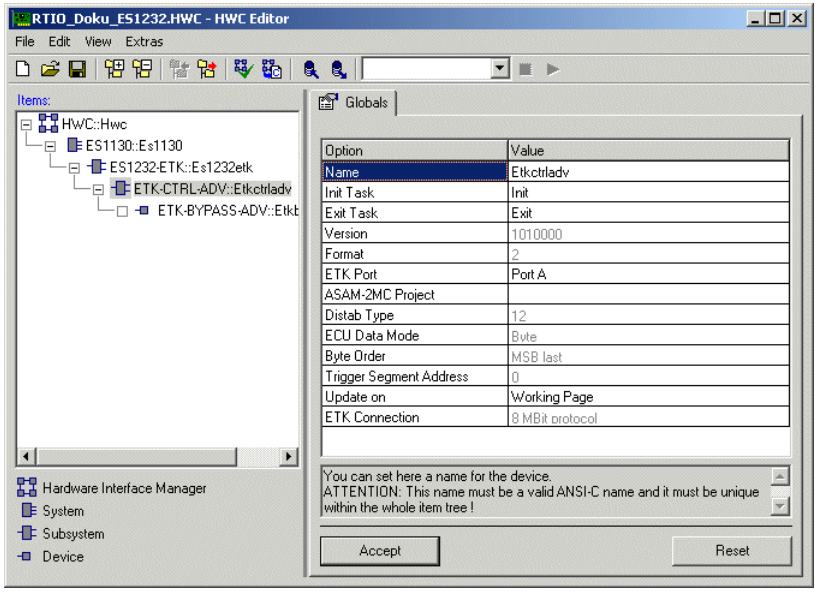


図 10-33 ETK-CTRL-ADV サブシステムの“Globals”タブ

ETK Port

ES1201（10.3.2 項を参照してください）の場合はこの行で ETK チャンネル（Port A、Port B）を選択しますが、ES1232 の場合は Port A しか使用できません。

ETK バイパスを行なうには、ASAM-MCD-2MC ディスクリプションファイルをデータベースに読み込んで作成される ASAM-MCD-2MC プロジェクト（AML V1.2）が必要です。155 ページの図 10-8 に、使用するプロジェクトを選択するためのダイアログボックスが示されています。

注記

AML V1.1 を使用した ASAM-MCD-2MC プロジェクトは読み込めません。そのようなプロジェクトをデバイスに割り当てようとすると、次のようなエラーメッセージが出力されます。
`Error: Incompatible version 0x<version> of QP_BLOB found in SOURCE '<sName>'. Expected version = 0x1. Selected ASAM-2MC Project is not suitable for the advanced ETK Controller ETK-CTRL-ADV ! Please select another one!`

バイパスについての記述は、ASAM-MCD-2MC ファイルの `IF_DATA ETK` セクションに含まれます。旧バージョンでは `IF_DATA ASAP1B_ETK` というセクション名が使用される場合がありましたが、これは ES1232 の ETK-CTRL-ADV サブシステムではサポートされていません。

各バイパスチャンネルごとに、チャンネル設定を定義する `QP_BLOB` があり、その例は 144 ページの「AML V1.1 以外のバージョン」の項に示されています。`TP_BLOB` には、以下のような一般的な設定が含まれます。

```
/begin TP_BLOB
   0x1000100     /* TP_BLOP version            */
   2             /* InterfaceSpeed  1=8MBit,   */
                 /*                 2=100MBit  */
   0x0           /* RstCfg parameter (MPC)     */
   /begin DISTAB_CFG  0xC 0x1 MSB_LAST 0x383000 0x0
    TRG_MOD 0xB7
   /end DISTAB_CFG
   CODE_CHK 0x0 0x0 0x0 0x0
   ETK_CFG 0xF 0xF0 0xFF 0x3 0xFD 0xEE 0xFF 0x1
   RESERVED 0x810000 0x8103F9 EXTERN 0x-1 0x-1 0x-1
      0x-1 0x-1
/end TP_BLOB
```

注記

`DISTAB_CFG` セクション内のパラメータは 138 ページの例を参照してください。また `CODE_CHK` および `RESERVED` セクションについては 154 ページを参照してください。

ES1232/ETK-CTRL-ADV の場合、ES1201、ES1231、および ES1232/ETK-CTRL-BAS とは異なり、すべての設定を ASAM-MCD-2MC ファイルから読み込む必要があり、HWC エディタからのマニュアル設定は行なえません。

ローカル BLD 定義についても同様で、HWC エディタでのマニュアル設定は行なえず、ASAM-MCD-2MC ファイルの MEASUREMENT セクション内において、必要な情報が KP_BLOB を使用して定義されている必要があります。

```
/begin MEASUREMENT log_uint8_0_A
   ""
   UBYTE
   ident
   1
   100
   0.0
   1.0
   READ_ONLY
   BIT_MASK 0x8
   ECU_ADDRESS 0xFD00
   /begin IF_DATA ASAP1B_Bypass
    /begin KP_BLOB
      BUFFER_OFFSET
         "log_uint8_0_offset_A.Model_Byp_A"
      /* no SOURCE_ID */
      BIT_OFFSET
         "log_uint8_0_bitOffset_A.Model_Byp_A"
      POSSIBLE_SOURCES 5
    /end KP_BLOB
   /end IF_DATA
/end MEASUREMENT
```

上の MEASUREMENT セクションには、158 ページの例と同じ内容が定義されています。

Distab Type

ECU とデータ交換に使用される DISTAB バージョンが表示されます。この情報は、ASAM-MCD-2MC プロジェクトから取得されます（138 ページを参照してください）。

- DISTAB12 では、2 バイトまでのデータを扱えます。
- DISTAB13 では、1、2、4、8 バイトのデータを扱えます。

ECU Data Mode

ECU プロセッサがデータメモリをアクセスする際のアクセスモード（Byte ／ Word）が表示されます。この項目は、DISTAB12 が使用される場合にのみ有効です。

この情報は ASAM-MCD-2MC プロジェクトから取得されます（138 ページを参照してください）。

Byte Order

ワードデータの格納フォーマット（MSB first ／ MSB last）が表示されます。この情報は ASAM-MCD-2MC プロジェクトから取得されます（138 ページを参照してください）。

Trigger Segment Address

DISTAB プロセスのトリガセグメントアドレスが表示され、編集は行なえません。この情報は ASAM-MCD-2MC プロジェクトから取得されるので（138 ページを参照してください）、あらかじめ適切なパラメータが定義されている必要があります。

トリガセグメントアドレスによって、バイパスチャンネル経由で行なわれるデータ転送に使用されるハードウェアトリガアドレスの位置が決まります。この位置は、DISTAB データ交換メソッド内で固定されています。

64 バイトのトリガセグメントの先頭アドレス（8 ビット幅または 16 ビット幅のバスアクセスの場合）は、64 で割り切れる偶数のアドレスでなければなりません。また、128 バイトのトリガセグメントの先頭アドレス（32 ビット幅のバスアクセスの場合）は、128 で割り切れるアドレスでなければなりません。

トリガセグメントのフォーマットは、DISTAB 12 および DISTAB 13 インターフェースに定義されています。

Update On

パラレル ETK を使用するバイパス通信において、シミュレーションボードと ECU 間のディスプレイテーブルの転送（139 ページの「バイパス通信（AML V1.1）」を参照してください）を、ETK のどのページ（`Working Page`、`Reference Page`、`Working & Reference Page`）で行うかを選択します。

シリアル ETK の場合は、すべてのディスプレイテーブルが RAM 領域内に位置するため、自動的に `RAM` ページが選択されます。この設定は変更できません。

ETK Connection

ボードの転送レート（8 Mbit/s または 100 Mbit/s）が表示されます。この値は ASAM-MCD-2MC ファイルから読み込まれたもので、ここで変更することはできません。

Update Offsets On

パラレルまたはシリアル ETK を使用するバイパス通信において、シミュレーションボードと ECU 間のバイパスオフセットの転送（139 ページの「バイパス通信（AML V1.1）」を参照してください）を、ETK のどのページ（`Working Page`、`Reference Page`、`Working & Reference Page`、`RAM`）で行うかを選択します。`RAM` はシリアル ETK の場合のみ選択できます。

シリアル ETK の場合はデフォルトで `RAM` ページが選択されます。

10.8.6 Globals（ETK-BYPASS-ADV デバイス）

ETK-BYPASS デバイスでは、ES1232/ETK-CTRL-ADV でバイパスを実行するために必要なすべての設定を行います。

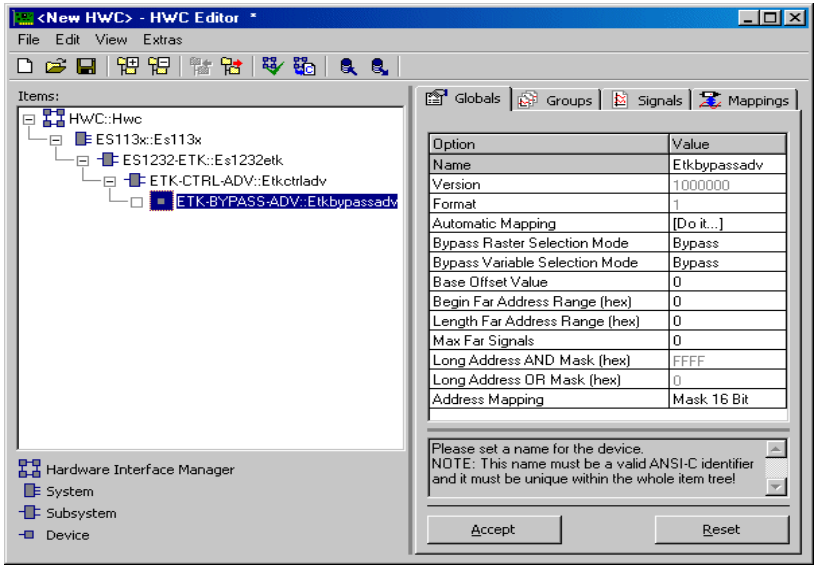


図 10-34 ETK-BYPASS デバイスの“Globals”タブ

Automatic Mapping

この項目については、173 ページの「Automatic Mapping」を参照してください。

Bypass Raster Selection Mode

“Groups”タブに、バイパスラスタおよび測定ラスタの両方を表示するか（`Bypass & Measuremnt`）、またはバイパスラスタのみを表示するか（`Bypass`：デフォルト）を指定します。

`Bypass & Measuremnt` を選択しておくと、INCA 用フックしか定義されていない場合でも、ASCET の実験環境において測定ラスタから取得されたシグナルを測定することができます。

この設定を変更した際、バイパスラスタ用に選択されているシグナルはすべてそのまま保持されますが、測定ラスタ用に選択されているシグナルは、`Bypass & Measuremnt` から `Bypass` に設定を変更すると、すべて選択が解除されます。

バイパスラスタと測定ラスタは同じように扱われますが、異なる点は、バイパスラスタが RTIO ドライバによってマスタアクセス用として初期化されるのに対し、測定ラスタはスレーブアクセス用として初期化される必要がある点です。しかし INCA は測定ラスタを常にマスタアクセスで使用するため、以下のような場合にエラーが発生する可能性があります。

1. すでに INCA による測定が行なわれている状態で ASCET バイパスを開始すると、RTIO ドライバは測定ラスタへのアクセス権が得られず、初期化を行なえません。
2. ASCET バイパス実験が行なわれている状態で INCA による測定が開始されると、INCA の方がより高いアクセス権を持つことになるため、ASCET から測定ラスタを奪い取ってしまうことになります。

どちらの場合も、エラーメッセージが表示されます。

Bypass Variable Selection Mode

10.3.3 項の「Bypass Variable Selection Mode」（159 ページ）を参照してください。

Base Offset Value

各バイパス出力データのバイトオフセット（0、2、4、または 8）を選択します。

バイトオフセットのデフォルト値は、DISTAB12 の場合は 0、DISTAB13 の場合は 8 です。

Begin Far Address Range (hex)

10.3.3 項の「Begin Far Address Range (hex)」（159 ページ）を参照してください。

Length Far Address Range (hex)

FAR アドレス領域のサイズを指定します。一般に、この設定は DISTAB 12 でのみ使用されます。

Max Far Signals

10.3.3 項の「Max Far Signals」（160 ページ）を参照してください。

Long Address AND / OR Mask (hex)

10.3.3 項の「Long Address AND / OR Mask (hex)」（160 ページ）を参照してください。

Address Mapping

10.3.3 項の「Address Mapping」（160 ページ）を参照してください。

10.8.7 Groups（ETK-BYPASS-ADV デバイス）

ETK-BYPASS-ADV デバイスのシグナルグループの数は、ASAM-MCD-2MC ファイルに定義されています。バイパスラスタと測定ラスタ用のシグナルグループは、10.8.6 項の“Globals”タブの「Bypass Raster Selection Mode」に設定された内容に応じて、自動的に生成または消去されます。

注記

“Bypass Raster Selection Mode”の設定が `Bypass & Measuremnt` から `Bypass` に変更されると、バイパスラスタ用に選択されているシグナルはそのまま残りますが、測定ラスタ用に選択されているシグナルはすべて消去されていしまいます。

1 つのバイパスラスタには 2 つのシグナルグループ（send-group と receive-group）が含まれ、測定ラスタには 1 つの receive-group のみが含まれます。

ECU と RTIO 間のデータ交換は、recieve シグナルグループの場合は標準の DISTAB メソッド、send シグナルグループの場合はバイパステーブルメソッドが使用されます。

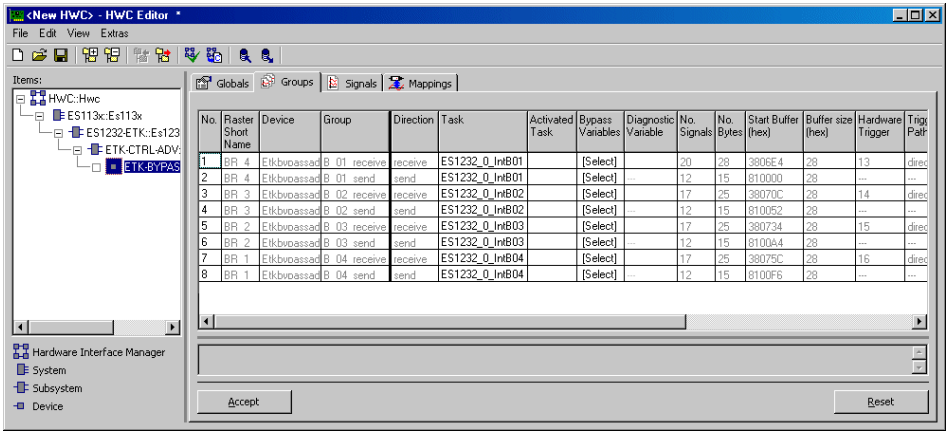


図 10-35 ETK-BYPASS-ADV デバイスの“Groups”タブ

Raster Short Name

ラスタのタイプ（`MT`： 測定ラスタ、`BT`： バイパスラスタ）と番号（優先度）が表示されます（例： `MT_12` 、`BT_31` など）。各シグナルグループは、デフォルトでは、優先度に従って降順にソートされて表示されます。

この値は ASAM-MCD-2MC プロジェクトから読み込まれるので、編集はできません。

Task

10.3.4 項の「Task」（161 ページ）を参照してください。

Activated Task

10.3.4 項の「ActivatedTask」（162 ページ）を参照してください。

Bypass Variables

10.5.4 項の「Bypass Variables」（186 ページ）を参照してください。

Diagnostic Variable

10.5.4 項の「Diagnostic Variable」（187 ページ）を参照してください。

No. Signals

各シグナルグループ用に選択されているシグナルの数が表示されます。この値は、シグナル選択を変更するたびに自動的に更新されます。

No. Bytes

各シグナルグループ用に選択されているシグナルの合計バイト数が表示されます。この値は、シグナル選択を変更するたびに自動的に更新されます。

Start Buffer (hex)

バイパス入力データおよび出力データ用に使用されるデータバッファのベースアドレスが表示されます。

この値は ASAM-MCD-2MC プロジェクトから読み込まれるので、編集はできません。

Buffer Size (hex)

バイパス入力データおよび出力データ用に使用されるデータバッファのサイズが表示されます。

この値は ASAM-MCD-2MC プロジェクトから読み込まれるので、編集はできません。

Hardware Trigger

receive シグナルグループに割り当てられたハードウェアトリガの数が表示されます。

この値は ASAM-MCD-2MC プロジェクトから読み込まれるので、編集はできません。

Trigger Path

receive シグナルグループ用のトリガが発生した時に行なわれるシグナル転送の方法が表示されます。

- `direct` - 1 つのラスタが直接ハードウェアトリガに割り当てられます。ECU がトリガアドレスへの書き込みを行なうと、直ちにデータの取得が行われます。トリガフラグの読み取りは行なわれません。
- `indirect` - 複数のラスタが 1 つのハードウェアトリガに割り当てられます。トリガを発生させたラスタを判定するために、各ラスタの“Trigger Flag Address”を読み取る必要があります。

充分な数のハードウェアトリガが使用可能であれば、直接転送が可能です。シリアル ETK の場合は、通常、間接転送しか行なえません。

send シグナルグループの場合、このカラムは使用されないため、“`---`”と表示されます。

Trigger Flag Address

このカラムは、“Trigger Path”で `indirect` が選択されているときにのみ使用され、正しいデータ取得を行なうために各ラスタごとに設定されたトリガフラグのアドレスが表示されます。この値は、ASAM-MCD-2MC プロジェクトから読み込まれ、ここで編集することはできません。

他の ETK バイパスデバイスで 8 Mbit/s 転送を行なう場合は、トリガ識別子（162 ページの「Trigger Id (hex)」）がこのトリガフラグに対応します。

“Trigger Path”が `direct` に設定されていると、ここには“`---`”と表示されます。

Start Pointer (hex)

receive シグナルグループについて、DISTAB メソッド用のポインタリストの先頭アドレスが表示されます。

この値は、ASAM-MCD-2MC プロジェクトから読み込まれ、ここで編集することはできません。

10.8.8 Signals（ETK-BYPASS-ADV デバイス）

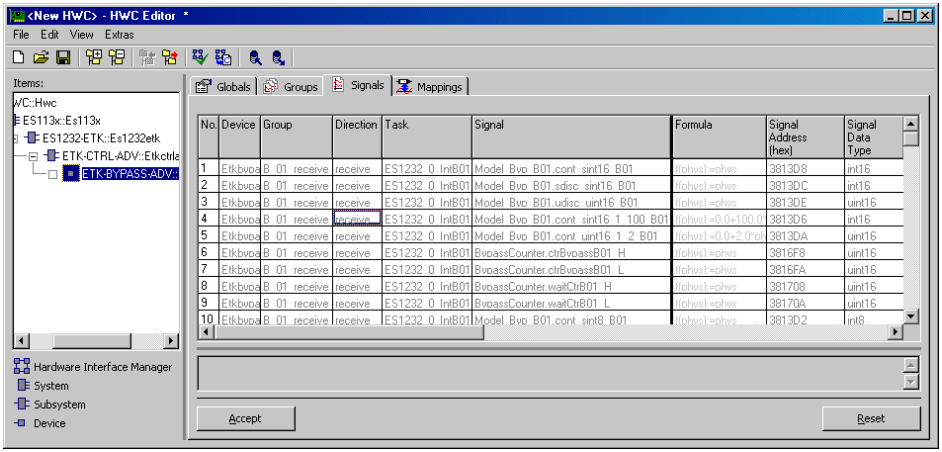


図 10-36 ETK-BYPASS デバイスの“Signals”タブ

このタブの内容は、10.3.5 項（163 ページ）および 10.7.5 項（197 ページ）を参照してください。

10.8.9 Mappings（ETK-BYPASS-ADV デバイス）

このタブ上の設定については、125 ページの 7.4.5 項を参照してください。

10.9 ES1300-AD

アナログ入力ボード ES1300 は、差動入力（最大 8 チャンネル）またはシングルエンド（最大 16 チャンネル）のアナログ信号を収集することができます。

10.9.1 Globals（ES1300-AD デバイス）

本項では、ES1300-AD デバイスのグローバルオプションについて説明します。

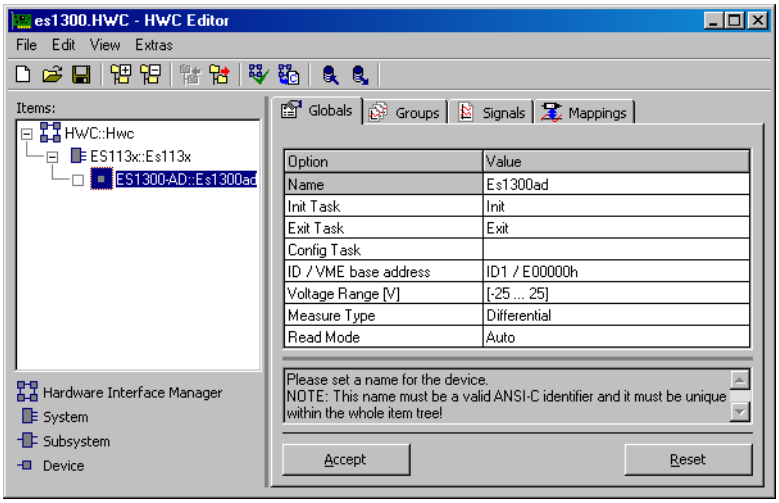


図 10-37 ES1300-AD デバイスの“Globals”タブ

Init Task

ES1300 を初期化するためのタスクを指定します（タイプ：Init、アプリケーションモード：active）。

Exit Task

実験を終了するときに ES1300 の終了処理を行なうタスクを指定します（タイプ：Init、アプリケーションモード：inactive）。

ID / VME base address

VME ベースアドレスの設定を行います。この設定はボード上のジャンパ設定と一致している必要があります。ES1300 の場合、4 種類の異なる VME ベースアドレス（`ID1/E00000h`、`ID2/E00100h`、`ID3/E00200h`、`ID4/E00300h`）から選択することができます。つまり、1 つの ES1000 システムで最大 4 枚の ES1300 を使用することができ、1 枚の ES1300 はベースアドレスから始まる 256 ワードのアドレススペースを占有します。

Voltage Range [V]

ES1300 の各チャンネルの入力電圧範囲を指定します。この設定は、ボード上のジャンパ設定と一致している必要があります。-25 ～ 25V、-50 ～ 50V、0 ～ 50V のいずれかを選択できます。

Measure Type

測定タイプとして、差動入力（最大 8 チャンネル）またはシングルエンド（最大 16 チャンネル）を選択します。この設定は、ボード上のジャンパ設定と一致している必要があります。

Read Mode

読み取りモードを指定します。ハードウェアドライバによる ASCET への A/D 測定値の転送を、A/D サンプリングの前と後のどちらに行うかを選択します。以下の設定が可能です。

- auto：
  ドライバは、前回のラスタの値を転送するか現在のラスタの値を転送するかを、ASCET がアクセスを 2 回行う間に経過した時間に基づいて決定します。
- wait for new values：
  ドライバは、現在のラスタからサンプル値が送られてきた後に転送を行います。
- get old values：
  ドライバは、サンプリングを行う前に前回のラスタの値を転送します。

10.9.2 Groups（ES1300-AD デバイス）

本項では、ES1300-AD デバイスのシグナルグループごとのオプションについて説明します。

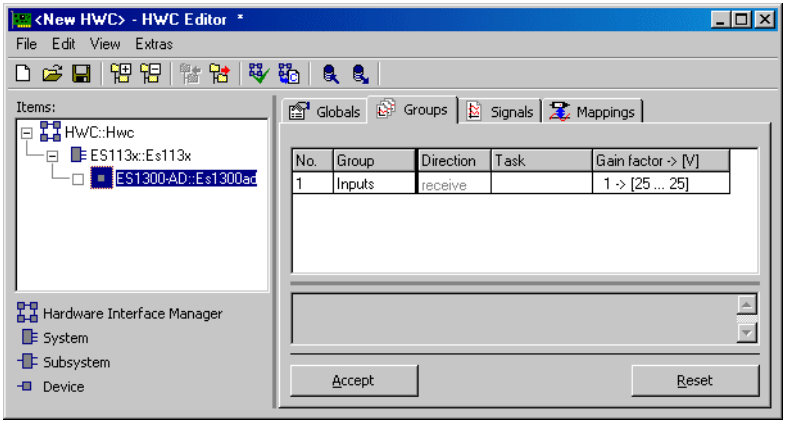


図 10-38 ES1300-AD デバイスの“Groups”タブ

Gain factor -> [V]

この項目で、アナログ入力信号用のプログラマブルゲインファクタ（1、5、10、50、100、500）を選択します。ゲインファクタにより、入力電圧の全域の中の限定された範囲を、より高い分解能で測定することが可能となります。

このカラムには、設定されたゲインファクタによって測定される電圧範囲も表示されます。この電圧範囲は、入力電圧範囲（215 ページの“Global”タブを参照してください）とゲインファクタから求められます。

10.9.3 Signals（ES1300-AD デバイス）

本項では、ES1300-AD デバイスの各シグナルごとのオプションについて説明します。

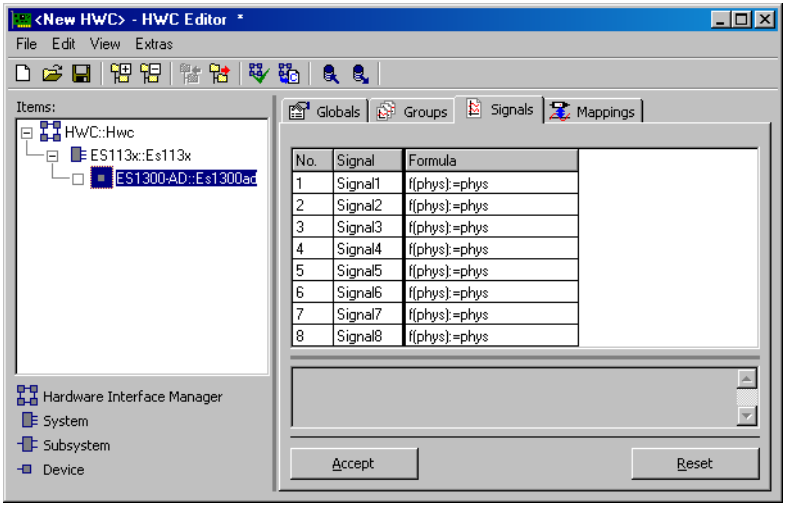


図 10-39 ES1300-AD デバイスの“Signals”タブ

No.

ES1300 で使用できるアナログ入力信号の最大数（8/16）は、“Globals”タブの `Measure Type` 設定（`differential/single-ended`）によって決まります。

10.9.4 Mappings（ES1300-AD デバイス）

このタブ上の設定は、すべてのデバイスについて共通です。125 ページの 7.4.5 を参照してください。

10.10 ES1301-AD

アナログ入力ボードの ES1301 を使えば、最大 7 点のアナログ信号を取り込むことができます。

10.10.1 Globals（ES1301-AD デバイス）

本項では、ES1301-AD デバイスのグローバルオプションについて説明します。

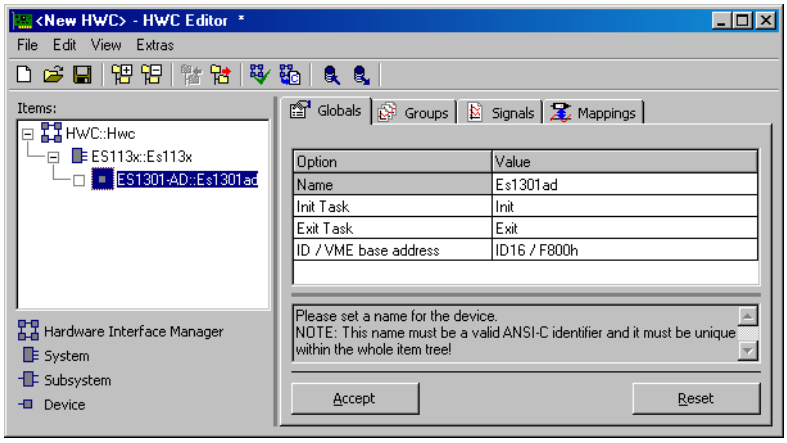


図 10-40 ES1301-AD デバイスの“Globals”タブ

Init Task

ES1301 を初期化するためのタスクを指定します（タイプ：Init、アプリケーションモード：active）。

Exit Task

実験を終了するときに ES1301 の終了処理を行なうタスクを指定します（タイプ：Init、アプリケーションモード：inactive）。

ID / VME base address

VME ベースアドレスの設定を行います。この設定は ES1301 にはんだストラップでセットされている設定と一致している必要があります。ES1301 の場合、5 種類の異なる VME ベースアドレス（**ID16/F800h**、**ID15/F000h**、**ID14/E800h**、**ID12/D800h**、**ID8/B800h**）から選択することができます。つまり、1 つの ES1000 システムで最大 5 枚の ES1301 を使用することができ、1 枚の ES1301 はベースアドレスから始まる 2048 ワードのアドレススペースを占有します。

10.10.2 Groups（ES1301-AD デバイス）

本項では、ES1301 デバイスのシグナルグループごとのオプションについて説明します。

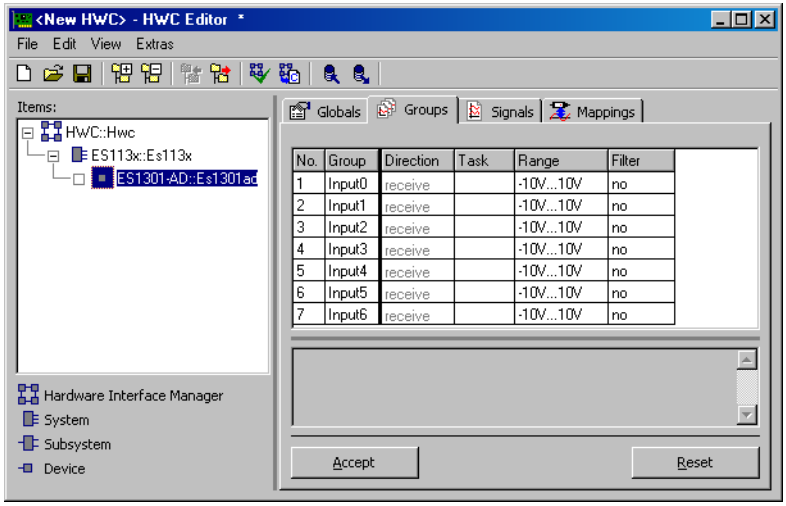


図 10-41 ES1301-AD デバイスの“Groups”タブ

Range

ES1301 の各チャンネルの入力電圧範囲を指定します。-10 ～ 10V、-5 ～ 5V、0 ～ 10V、および 0 ～ 5V の入力電圧範囲から選択することができます。入力電圧範囲を選択するための設定はソフトウェアにより行われるので、ジャンパ設定は不要です。

Filter

各チャンネルについて 2 次のローパスフィルタを設定することができます。フィルタを使用する場合、100Hz、200Hz、500Hz、1KHz、2KHz、5KHz、および 10KHz のいずれかの周波数を選択できます。

10.10.3 Signals（ES1301-AD デバイス）

本項では、ES1301-AD デバイスの各シグナルごとのオプションについて説明します。

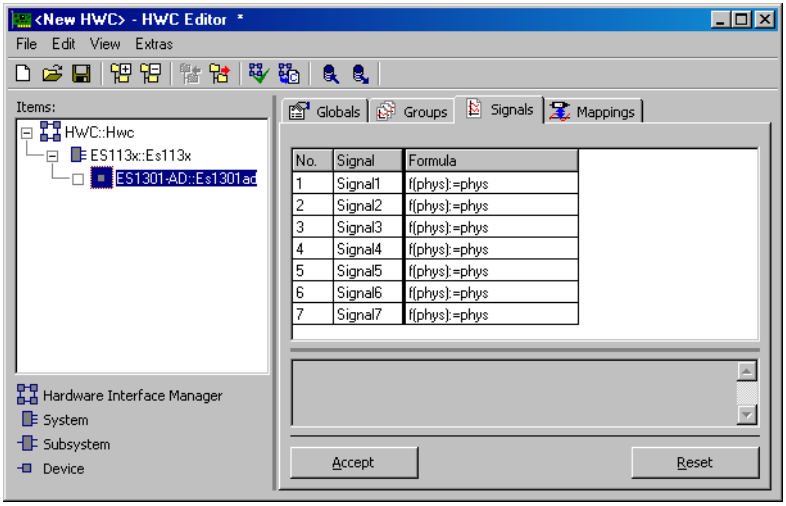


図 10-42 ES1301-AD デバイスの“Signals”タブ

ES1301-AD デバイスの“Signals”タブには、このデバイス固有の項目はありません。

10.10.4 Mappings（ES1301-AD デバイス）

このタブ上の設定は、すべてのデバイスについて共通です。125 ページの 7.4.5 を参照してください。

10.11 ES1303-AD

ES1303 A/D ボードは、最大 16 チャンネルのアナログ信号を、高速サンプリングレート（最大 100 kHz、信号が複数の場合は 50 kHz まで可能）において高分解能の読み取りを行うことができます。ES1303 は、1 システム内で同時に 4 枚まで使用できます。

信号の取り込みの開始と終了を 2 つのトリガ入力で制御でき、その際、連続測定またはシングルショット測定（指定した 1 つの値を測定）を行なえます。詳しくは、ES1303 のユーザーズガイドを参照してください。

16 の測定チャンネルは、個々に ASCET-RP または INCA で使用できます。つまり、両方のアプリケーションで 1 枚のボートを共有することができます。ただし、ES1303 のコンフィギュレーションがすでに INCA から設定されていた場合、

ASCET-RP でその設定を上書きすることはできず、設定を行おうとするとエラーメッセージが表示されます。また 2 つのデジタルチャンネルは、いずれか一方のアプリケーションからしか使用できません。

10.11.1 Globals（ES1303-AD デバイス）

本項では、ES1303-AD デバイスのグローバルオプションについて説明します。

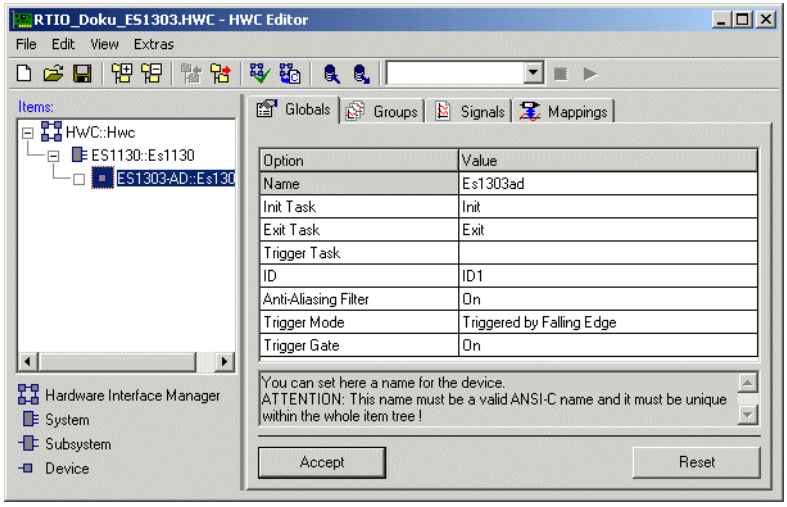


図 10-43 ES1303-AD デバイスの“Globals”タブ

Init Task

ES1303 を初期化するためのタスクを指定します（タイプ：Init、アプリケーションモード：active）。

Exit Task

実験を終了するときに ES1303 の終了処理を行なうタスクを指定します（タイプ：Init、アプリケーションモード：inactive）。

IRQ Handler Task

シグナルチャンネルを割り込みモードで測定する場合、ここに IRQ ハンドラタスク（タイプ：Software、アプリケーションモード：active）を指定します。

このタスクは、ASCET プロジェクトエディタ上のタスクリスト内に“Software”タスクとして定義し、“Max. number of Activations”フィールドの値は 2 以上（最大は 50）に設定してください。

注記

このタスクは、デバイスとそれに対応するハードウェアドライバ専用のものである必要があり、他のデバイスやユーザープロセスと共有することはできません。同じタイプのデバイス（たとえば 2 枚の ES1303）であっても、同じタスクの共有はできません。

この項目は、ハードウェアトリガモード（「HW Trigger Mode」を参照してください）が `off` に設定されている場合、無効となります。

ID

ボード番号を入力します。

注記

ES1303 は、システムがオンになったときにハードウェアマネージャにより ID 番号およびフリーアドレス領域が自動的に割り当てられる、“Auto-ID”ボードと呼ばれるタイプのボードです。同じタイプのボードが複数存在する場合には、左から右の順に番号が付けられます。つまり、一番左のボードの番号が 1 になり、その次の番号は 2 になります。1 つの実験システムでは最大 4 枚のボードを使用することができ、4 種類の ID を指定できます。

Anti-Aliasing Filter

アンチエイリアシングフィルタをオン（`On`）またはオフ（`Off`）にします。このフィルタによって入力信号の周波数域が限定され、サンプリングの一般原則（「サンプリングレートは入力信号の周波数の 2 倍以上でなければならない」という条件）が守られます。

HWC エディタからのフィルタの ON/OFF は全チャンネル同時に行われますが、ハードウェア自体には、それぞれ 4 つのチャンネルを含む 4 つのチャンネルグループ（ch1 ～ 4、ch 5 ～ 8、ch9 ～ 12、ch13 ～ 16）ごとに切り替えを行なう機能があります。そのため、ASCET と INCA で同じグループ内のチャンネルを異なるフィルタ設定でアクセスを行なうと、問題が生じます。たとえば、ASCET がチャンネル 1 と 2 をフィルタなしでアクセスし、INCA がチャンネル 3 と 4 をフィルタを使用してアクセスした場合、同じグループに属するそれらのチャンネルは同じ設定である必要があるため、矛盾が発生してしまいます。このような場合は、エラーメッセージが表示されるので、たとえば INCA でチャンネル 3 と 4 の代わりに 5 と 6 を使用するように変更してください。

アンチエイリアシングフィルタは、100 kHz のサンプリングレート用に最適化されています。このフィルタが使用されない場合は、代わりにローパスフィルタ（カットオフ周波数 255 kHz）が使用されます。

HW Trigger Mode

トリガ測定を開始する際のトリガモードを指定します。以下のモードから選択できます。

- `Off`：トリガ信号を使用しません。
- `Rising Edge`：トリガ信号の最初の立ち上がりエッジで測定を開始します。『ES1303 ユーザーズガイド』のモード 3 についての説明を参照してください。
- `Falling Edge`：トリガ信号の最初の立ち下がりエッジで測定を開始します。『ES1303 ユーザーズガイド』のモード 4 についての説明を参照してください。

“Groups”タブで“IRQ”のカラムが `No` に設定されているシグナルグループ（225 ページの「IRQ」を参照してください）については、トリガモード設定は無効です。

HW Trigger Gate

トリガの生成のために第 2 のトリガ信号を使用する（`ON`）かしない（`OFF`）かを指定します。

トリガゲートが ON に設定されていると、トリガモードに基づいて 1 番目のトリガ信号の最初のエッジが検出された時、2 番目のトリガ信号が High になっている場合にのみ測定が開始されます。トリガモードが OFF になっていれば、1 番目のトリガ信号によってのみ測定が開始されます。以下の表は、測定が開始される条件をまとめたものです。

HW Trigger Mode（トリガモード）	**HW Trigger Gate（トリガゲート）**	**トリガ信号 1**	**トリガ信号 2**
`Rising Edge`	`On`	立ち上がりエッジ	High
`Rising Edge`	`Off`	立ち上がりエッジ	不使用
`Falling Edge`	`On`	立ち下がりエッジ	High
`Falling Edge`	`Off`	立ち下がりエッジ	不使用

“Groups”タブで“IRQ”のカラムが `No` に設定されているシグナルグループ（225 ページの「IRQ」を参照してください）については、トリガゲート設定は無効です。

10.11.2 Groups（ES1303-AD デバイス）

本項では、ES1303-AD デバイスのシグナルグループごとのオプションについて説明します。

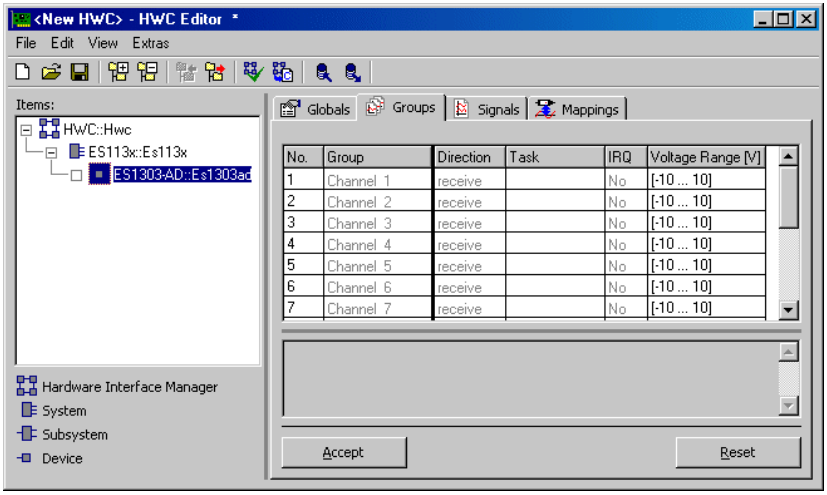


図 10-44 ES1303-AD デバイスの“Groups”タブ

16 個のシグナルグループが定義されていて、各シグナルグループごとに 1 つのシグナルが割り当てられています。

注記

ES1303 は、将来的には 8 チャンネルのみをサポートするようになります。HWC エディタ上で 8 チャンネル以上のチャンネルが定義されていて、実際のボードには 8 チャンネルしか搭載されていない場合、モデルの実行開始時にエラーメッセージが表示されます。その場合、ES1303 は無視されます。

Task

シグナルの受信タスクを指定します。複数のラスタから受信を行なう場合は複数のタスクを指定します。

受信が割り込みモードで行なわれる場合は、このカラムの内容はリセットされ、入力もできません。

IRQ

シグナルの受信を割り込みモード（Yes）で行なうかどうかを指定します。この項目は、ハードウェアトリガモード（「HW Trigger Mode」を参照してください）が `off` に設定されている場合、無効となります。

通常の受信処理（No）の場合は、シャノンの定理に従い、受信タスクのサンプリング周波数を、測定する入力信号の 2 倍以上にする必要があります（『ES1303 ユーザーズガイド』の 3.1.2 項を参照してください）。固定ラスタで測定されない信号や散発的にしか発生しない信号は、割り込みモードを使用し、発生した信号が確実に処理されるようにしてください。

各シグナルグループは個々に異なる設定を行なえます。“Trigger Mode”と“Trigger Gate”は、割り込みモードで受信されるすべてのシグナルグループに適用されます。

空のシグナルグループ、つまり IRQ もタスクも指定されていないグループについては、プロセスが生成されません。ASCET モニタウィンドウにワーニンメッセージが表示されます。

Voltage Range [V]

測定チャンネルの入力電圧範囲（-10 ～ 10V または -60 ～ 60V）を選択します。

10.11.3 Signals（ES1303-AD デバイス）

本項では、ES1303-AD デバイスの各シグナルごとのオプションについて説明します。

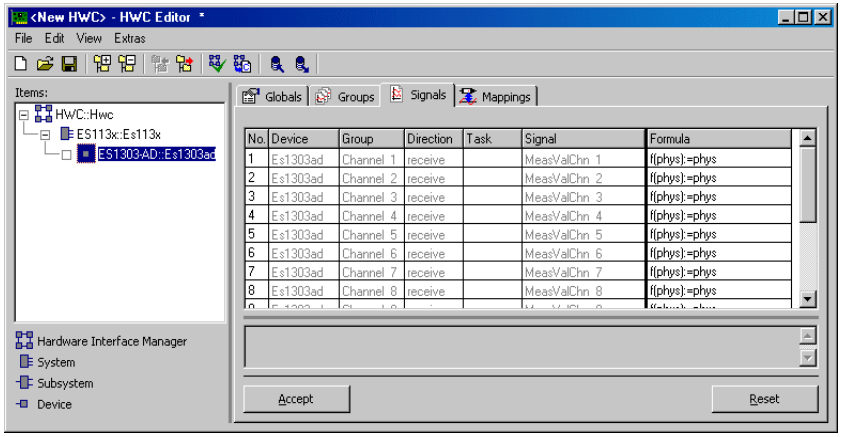


図 10-45 ES1303-AD デバイスの“Signals”タブ

ES1300（10.9.3 項）の場合とは異なり、この“Signals”タブでは変換式しか編集できません。その他のカラムについては、7.4.4 項に説明されています。

10.11.4 Mappings（ES1303-AD デバイス）

このタブ上の設定は、すべてのデバイスについて共通です。125 ページの 7.4.5 を参照してください。

10.12 ES1310-DA

アナログ出力ボードの ES1310 を使えば、8 点のアナログ信号を出力することができます。

10.12.1 Globals（ES1310-DA デバイス）

本項では、ES1310-DA デバイスのグローバルオプションについて説明します。

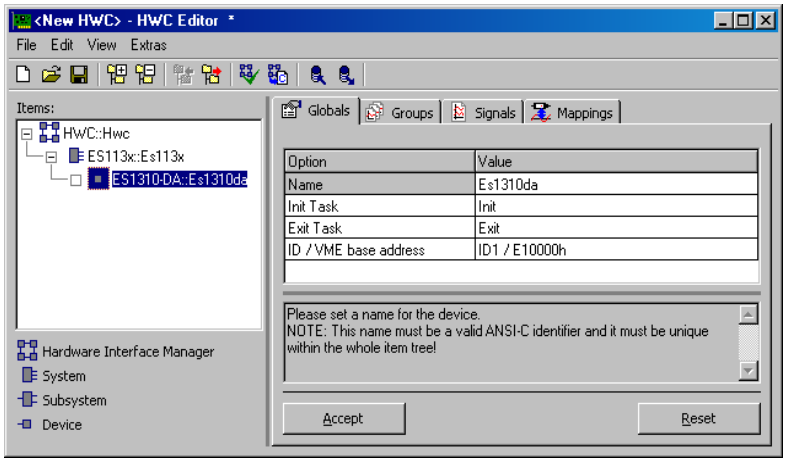


図 10-46 ES1310-DA デバイスの“Globals”タブ

Init Task

ES1310 を初期化するためのタスクを指定します（タイプ：Init、アプリケーションモード：active）。

Exit Task

実験を終了するときに ES1310 の終了処理を行なうタスクを指定します（タイプ：Init、アプリケーションモード：inactive）。

ID / VME base address

この行では、VME ベースアドレスの設定を行います。この設定はボード上のジャンパ設定と一致している必要があります。ES1310 の場合、2 種類の異なる VME ベースアドレス（`ID1/E10000h`、`ID2/E20000h`）から選択することができます。つまり、1 枚の ES1000 システムで最大 2 枚の ES1310 を稼働させることができ、1 枚の ES1310 はベースアドレスから始まる 1024 ワードのアドレススペースを占有します。

10.12.2 Groups（ES1310-DA デバイス）

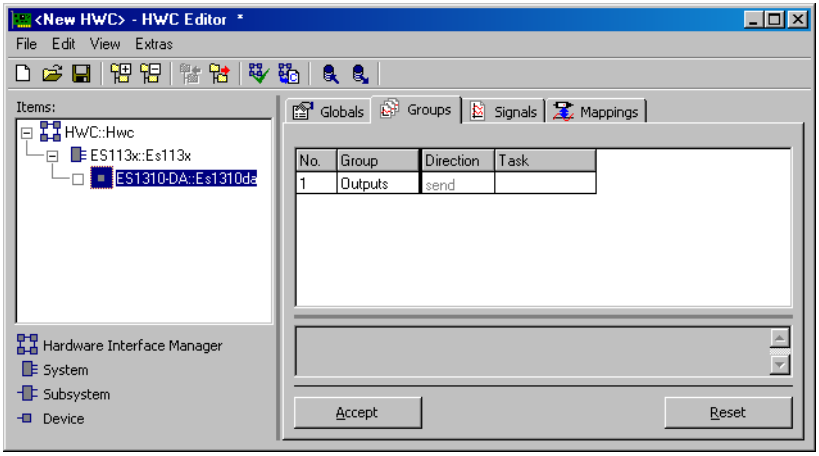


図 10-47 ES1310-DA デバイスの“Groups”タブ

ES1310-DA デバイスの“Groups”タブには、このデバイス固有の項目はありません。

10.12.3 Signals（ES1310-DA デバイス）

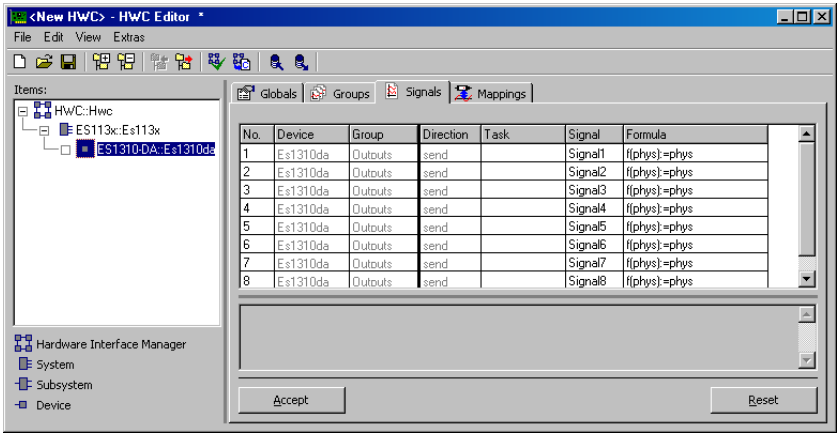


図 10-48 ES1310-DA デバイスの“Signals”タブ

ES1310-DA デバイスの“Signals”タブには、このデバイス固有の項目はありません。

10.12.4 Mappings（ES1310-DA デバイス）

このタブ上の設定は、すべてのデバイスについて共通です。125 ページの 7.4.5 を参照してください。

10.13 ES1320-CB（DIO）

デジタル入出力ボードの ES1320（DIO）を使えば、20 点のデジタル信号の測定や出力を行なうことができます。ES1320 は 2 つの DIO ピギーバックボードを搭載したキャリアボード（CB）です。

HWC エディタのアイテムリストに組み込む際は、キャリアボードはサブシステムとして追加し、DIO ピギーバックボードはデバイスとして追加します。

10.13.1 Globals（ES1320-CB サブシステム）

本項では、ES1320-CB サブシステムのグローバルオプションについて説明します。

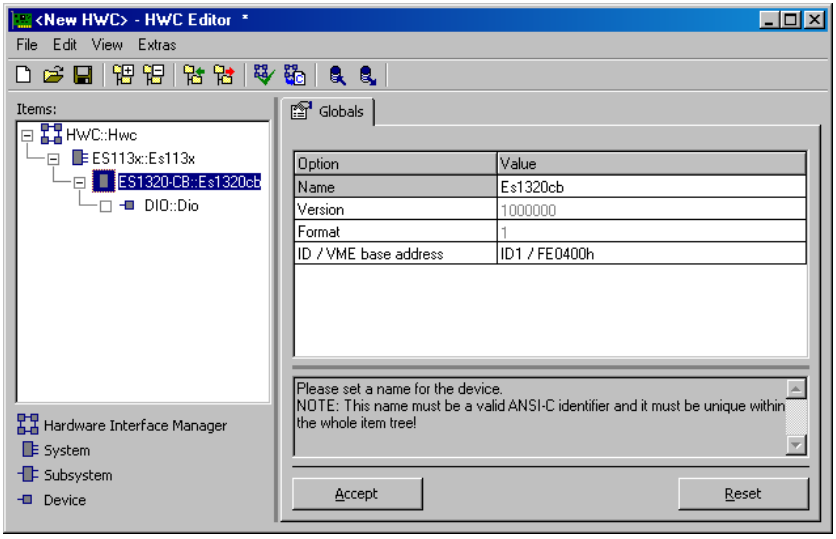


図 10-49 ES1320-CB サブシステムの“Globals”タブ

ID / VME base address

この行では、VME ベースアドレスの設定を行います。この設定はボード上のジャンパ設定と一致してしている必要があります。ES1320 の場合、8 種類の異なる VME ベースアドレス（`ID1/FE0400h`、`ID2/FE0C00h`、`ID3/FE1400h`、`ID4/FE1C00h`、`ID5/FE2400h`、`ID6/FE2C00h`、`ID7/FE3400h`、`ID8/FE3C00h`、`ID9/FE4400h`、`ID10/FE4C00h`、`ID11/FE5400h`、`ID12/FE5C00h`、`ID13/FE6400h`、`ID14/FE6C00h`、`ID15/FE7400h`、`ID16/FE7C00h`）から

選択することができます。つまり、1 つの ES1000 システムで最大 8 枚の ES1320 を稼働させることができ、1 枚の ES1320 はベースアドレスから始まる 256 ワードのアドレススペースを占有します。

10.13.2 Globals（DIO デバイス）

本項では、DIO デバイスのグローバルオプションについて説明します。

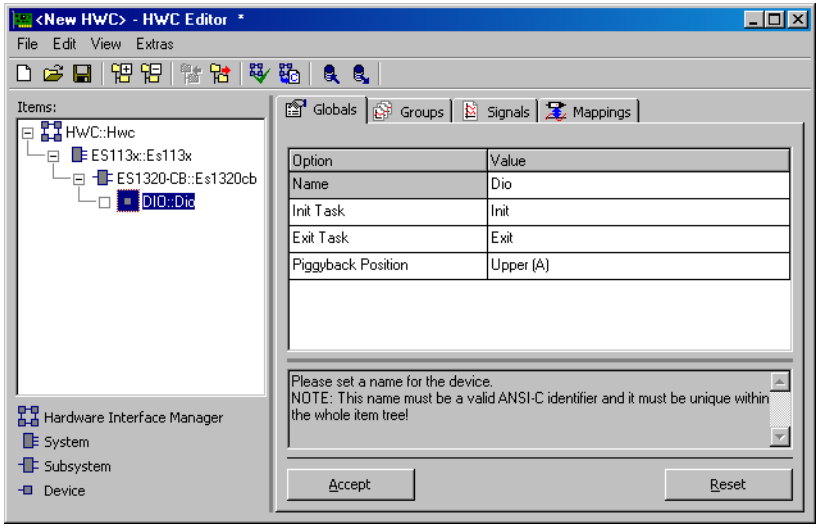


図 10-50 DIO デバイスの“Globals”タブ

Init Task

ES1320 を初期化するためのタスクを指定します（タイプ：Init、アプリケーションモード：active）。

Exit Task

実験を終了するときに ES1320 の終了処理を行なうタスクを指定します（タイプ：Init、アプリケーションモード：inactive）。

Piggyback Position

デジタルインターフェースとして使用する DIO ピギーバックボード（`Upper(A)` または `Lower(B)`）を選択します。

10.13.3 Groups（DIO デバイス）

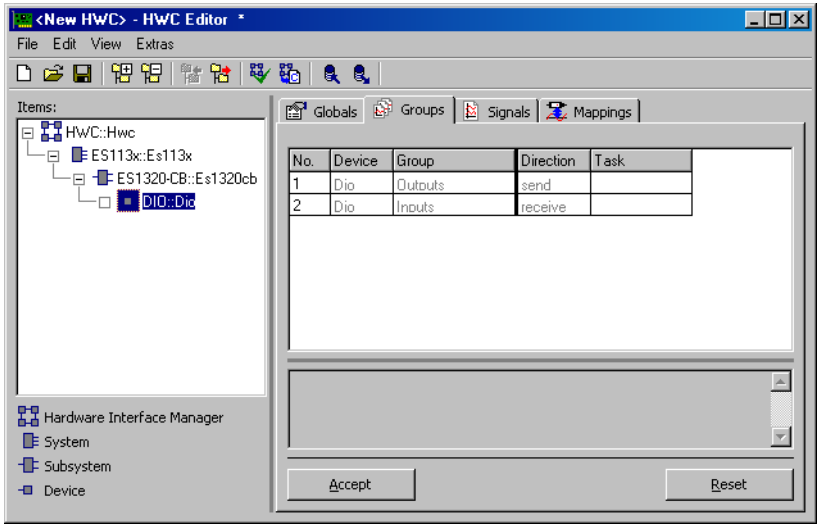


図 10-51 DIO デバイスの“Groups”タブ

DIO デバイスの“Groups”タブには、このデバイス固有の項目はありません。

10.13.4 Signals（DIO デバイス）

本項では、DIO デバイスの各シグナルごとのオプションについて説明します。

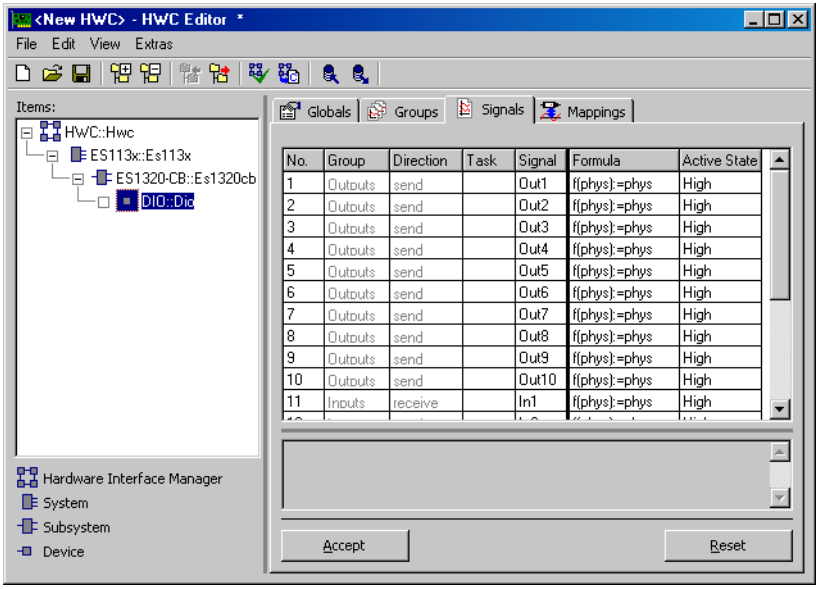


図 10-52 DIO デバイスの“Signals”タブ

Active State

信号が反転信号（`Active Low`）かそうでない（`Active High`）かを指定します。

10.13.5 Mappings（DIO デバイス）

このタブ上の設定は、すべてのデバイスについて共通です。125 ページの 7.4.5 を参照してください。

10.14 ES1325-DIO

デジタル入／出力ボード ES1325（DIO）は、デジタル信号の取り込みと出力を、それぞれ 16 点のチャンネルで行うことができます。また 2 つのトリガ入力により、信号の取り込みと出力を制御することも可能です。1 枚の ES1130 または ES1135 について、最大 4 枚の ES1325 ボードを使用できます。RTIO においては、1 枚の ES1325 ボードについて、Input デバイス、Output デバイス、LED デバイスを、それぞれ最大 1 つずつまで組み込むことができます。

10.14.1 Globals（ES1325-DIO サブシステム）

本項では、ES1325-DIO サブシステムの一般的なオプションについて説明します。

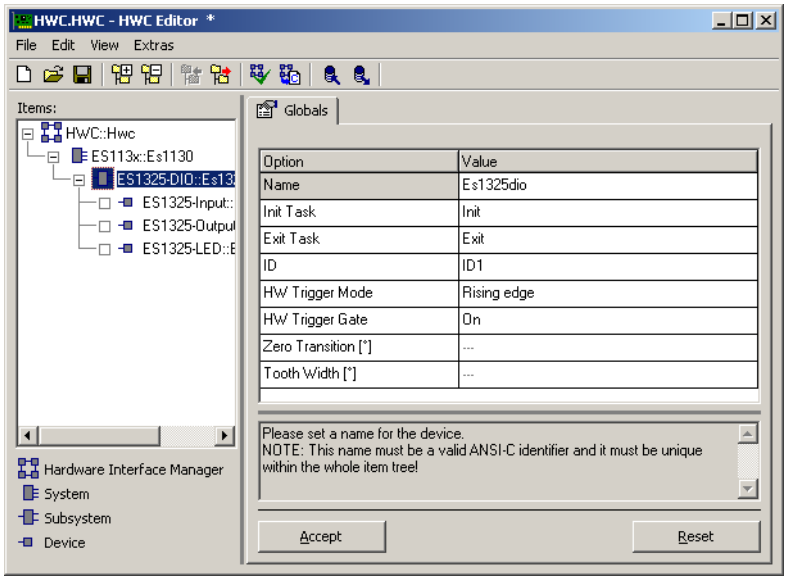


図 10-53 ES1325-DIO サブシステムの“Globals”タブ

ID

ここでは、ボード番号を示す ID をボードに割り当てます。

注記

ES1325 ボードは、システムがオンになったときにハードウェアマネージャにより ID 番号およびフリーアドレス領域が自動的に割り当てられる、“Auto-ID” ボードと呼ばれるタイプのボードです。同じタイプのボードが複数（たとえば ES!325 ボードが 2 枚）存在する場合には、左から右の順に番号が付けられます。つまり、一番左のボードの番号が 1 になり、その次の番号は 2 になります。1 つの実験システムでは最大 4 枚のボードを使用することができ、4 種類の ID を指定できます。

HW Trigger Mode

トリガ測定を開始する際のトリガモードを指定します。以下のモードから選択できます。このオプションは、“HW Trigger Gate” が **On** に設定されている場合にのみ有効です（235 ページの「HW Trigger Gate」を参照してください）。

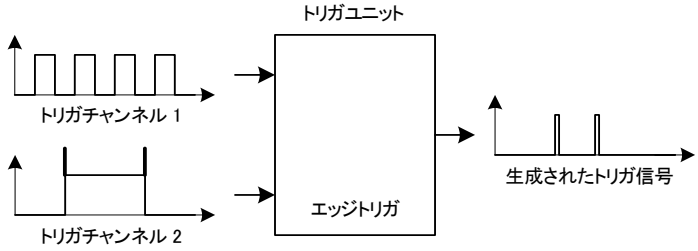
以下のトリガモードを選択できます。

- **Off**

  トリガ入力信号を使用しません。

- **Rising edge**

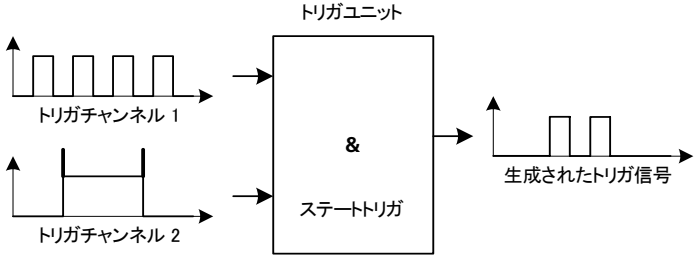
  トリガ入力 1 の信号の立ち上がりエッジにおいて、デジタル入力チャンネルからの信号の取り込みや、デジタル出力チャンネルへの信号出力（信号生成）が行われます。以下の図は、“HW Trigger Gate”がオンになっている場合のトリガ信号の生成パターンを示しています。



- **Signal state**

  トリガ入力 1 の信号の立ち上がりエッジによってデジタル入出力信号の取り込みや出力が開始し、立ち下がりエッジで終了します。つまり、トリガ入力 1 がアクティブレベルである間、信号入出力が行われます。

  以下の図は、“HW Trigger Gate”がオンになっている場合のトリガ信号の生成パターンを示しています。



  “Signal state”トリガモードでは、オプション設定により、信号入出力処理のリクエストのタイミングを遅延させることができます（242 ページの「Signals」を参照してください）。ただしこの遅延時間はトリガ信号のホールド時間よりも短くする必要があります。ホールド時間以上に設定すると、入出力処理が開始されなくなってしまいます。

- **Angle based**

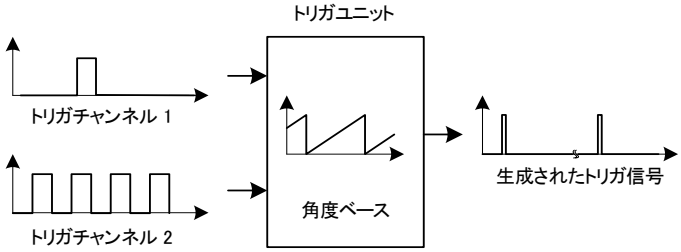
  トリガ入力 1 とトリガ入力 2 によって、クランクシャフトの回転角を算出します。

  各トリガ入力には、以下のクランクシャフト信号を接続します。

– トリガ入力 1：ゼロトランジション（角度リセット信号）

– トリガ入力 2：歯幅

トリガ入力 2 においては、1 つの PWM 周期が 1 つの角度セグメントに相当します。2 つのトリガ入力を組み合わせて PWM 周期をカウントすることにより、角度位置、つまり現在の回転角が算出されます。



このトリガモードにおいては、トリガユニット上で、2 つのトリガ入力によって角度カウンタがカウントされます。

トリガ入力 1 の立ち上がりエッジでカウンタがゼロにリセットされ、トリガチャンネル 2 の立ち下がりエッジでカウンタ値が 1 だけ大きくなります。これにより、一定の角度セグメント（例えば、歯幅 6°）ごとにトリガ入力 2 の立ち下がりエッジが検出され、さらにクランクシャフトが 1 回転するたびに（つまり 0° へのトランジションの時に）トリガ入力 1 の立ち上がりエッジが検出されるようにすれば、現在のクランク角が取得できます。

正しい動作を保証するためには、トリガ入力 2 の信号は、連続する等間隔の角度セグメントを示す必要があり、トリガ入力 1 の信号の立ち上がりエッジによって、ゼロトランジションのタイミングが示される必要があります。また、トリガ 1 の立ち下がりエッジは、トリガ入力 2（角度セグメント信号）の立ち下がりエッジ後、所定のホールド時間の後に発生する必要があります。

上記の図に示されているように、トリガモード **`Rising edge`** および **`Signal state`** の場合、トリガチャンネル 2 をトリガゲートとして定義することもできます（235 ページの「HW Trigger Gate」を参照してください）。

HW Trigger Gate

トリガ入力のエッジ検出を行うモードを設定します。このパラメータがオンになっていると、トリガ入力 2 がゲート信号として使用されます。つまり、トリガ入力 2 がイネーブル状態（=TTL High）である時にのみトリガ入力 1 の信号が有効になります。

以下のモードを使用できます。

- **`Off`**

  トリガ入力 1 において、“HW Trigger Mode”で指定されたエッジが検出されるたびに、デジタル入出力処理が行われます。

- **On**

  トリガ入力 2 が HIGH レベルになっている間のみ、トリガ入力 1 の所定のエッジによってデジタル入出力処理が行われます。

トリガモードとして **Angle based** が選択されている場合（233 ページの「HW Trigger Mode」を参照してください）、“HW Trigger Gate”パラメータは自動的に“**On**”になっています。また、トリガモードが **Off** になっている場合は、“HW Trigger Gate”パラメータを編集することはできません。

Zero Transition [*]

トリガ入力 1 のゼロトランジション信号について定義します。

分解能		歯幅
値の範囲	最小	2 ×歯幅
	最大	32767 ×歯幅
クランク信号としての典型的な値		360° または 720°

このオプションはトリガモードが **Angle based** である場合にのみ使用でき（233 ページの「HW Trigger Mode」を参照してください）、上記の値の範囲内で編集できます。

Tooth Width [*]

トリガ入力 2 について、1 つの PWM 周期分に相当する歯幅を定義します。

分解能		0.5°
値の範囲	最小	0.5°
	最大	720°
クランク信号としての典型的な値		6°

このオプションはトリガモードが **Angle based** である場合にのみ使用でき（233 ページの「HW Trigger Mode」を参照してください）、上記の値の範囲内で編集できます。

10.14.2 Globals（ES1325-Input デバイス）

本項では、Input デバイスの一般的なオプションについて説明します。

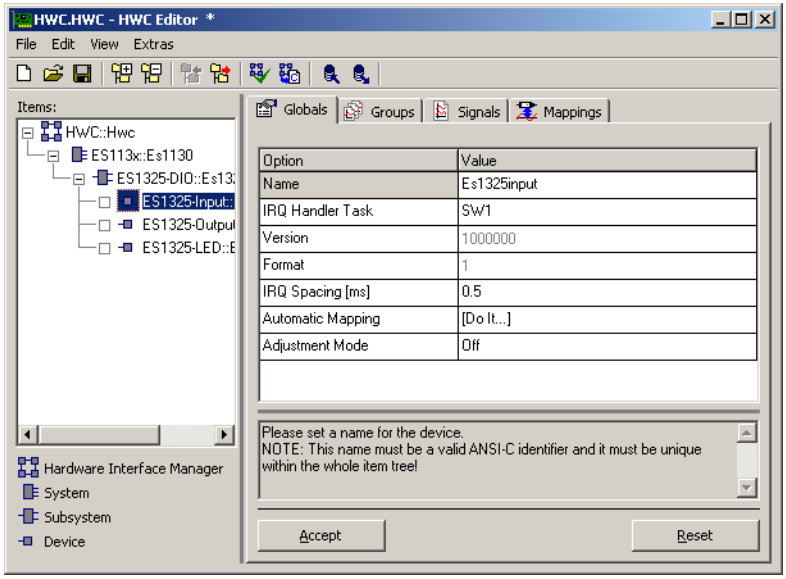


図 10-54 ES1325-DIO Input デバイスの“Globals”タブ

IRQ Handler Task

各シグナルチャンネルを割り込みモードで評価する場合は、この“IRQ Handler Task”（タイプ：Software、アプリケーションモード：active）が必要です。

このタスクは、ASCET プロジェクトエディタ上のタスクリスト内に“Software”タスクとして定義し、“Max. number of Activations”フィールドの値は 2 以上（最大は 50）に設定してください。

注記

このタスクは、アイテムとそれに対応するハードウェアドライバごとに**個別**のものを指定する必要があり、ここで指定されたタスクを他のアイテムやユーザープロセスで共有することはできません。また同じタイプのアイテム（たとえば 2 枚の ES1325）であっても、同じタスクを共有することはできません。

このオプションは、トリガモードが **Off** になっている場合は、使用されません（233 ページの「HW Trigger Mode」を参照してください）。

IRQ Spacing [ms]

ES1325 により生成される割込み要求の最小間隔を定義します。

分解能		0.1 ms
値の範囲	最小	0 ms
	最大	25.5 ms

このオプションは、トリガモードが **Off** 以外のモードになっている場合（233 ページの「HW Trigger Mode」を参照してください）に、上記の値の範囲内で編集できます。

Automatic Mapping

このオプションで、シグナルを ASCET メッセージに自動的に割り当てます。

詳しくは 173 ページの「Automatic Mapping」を参照してください。

Adjustment Mode

「調整モード」を有効にすることにより、オンライン時、つまり ASCET モデル実行時に、信号収集に影響を与えることができます。

以下のモードから選択できます。

- **Off**

  調整モードは無効になります。各入力チャンネル（“ChXX”）ごとに 1 つの "ChXXMsr” 測定グループが表示されます。

- **On**

  調整モードが有効になります。各“ChXX”入力チャンネルごとに、“ChXXMsr”測定グループ以外に、“ChXXAdj”という「オンライン調整グループ」も表示されます。この“ChXXAdj”グループには、オンライン時に“Groups”タブの“Signals”列で調整できるコンフィギュレーションパラメータも含まれています。

Adjustment Mode は、HW Trigger Mode が **On** になっている場合にのみ選択できます（233 ページの「HW Trigger Mode」を参照してください）。

10.14.3 Groups（ES1325-Input デバイス）

本項では、Input デバイスの各シグナルグループ用オプションについて説明します。

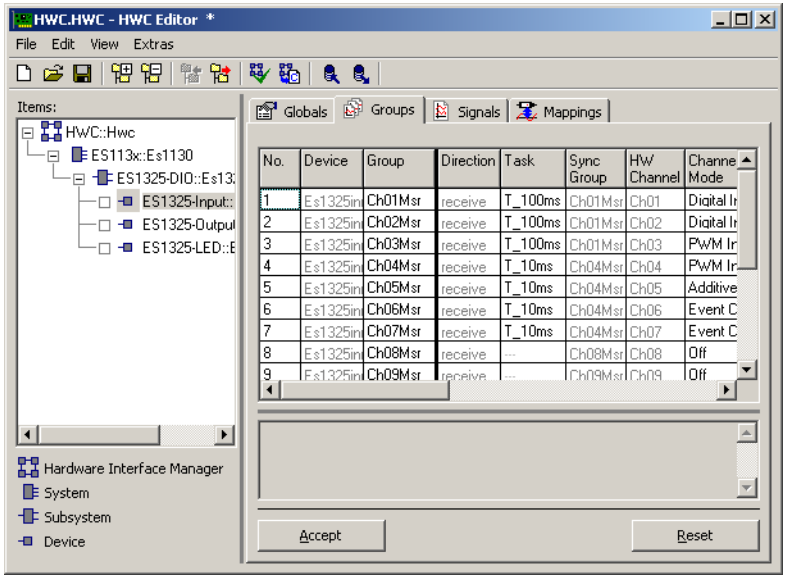


図 10-55 ES1325-DIO Input デバイスの“Groups”タブ

Sync Group

同じタスク（またはタスクグループ）によって同調して転送されるシグナルグループは、1 つのプロセスに割り当てられます。このプロセスには、この列に表示されているグループ名と同じ名前が付きます。

シグナルグループに複数のタスクが選択されている場合、“Task”フィールドの値が同じシグナルグループのプロセスのみ、1 つの共通プロセスに割り当てることができます。

このプロセスに割り当てられたシグナルグループは、同調して転送されます。

Hardware Channel

ES1325 の入力チャンネル（Ch01 ～ Ch16）

Channel Mode

ES1325 の各入力チャンネルについて、デジタル入力信号をどのように評価するかを指定します。

- Off

  デジタル入力信号を評価しません。
- Digital Input

  リクエストされた時点のデジタル入力信号のステートを評価します。
- PWM Input

  リクエストされるまでの時点において、最後に取り込まれた 1 周期分の完全な PWM 信号についての情報を評価します。
- Additive Time

  リクエストされたインターバル内で、信号のアクティブフェーズの時間を合計します。
- Event Counter

  リクエストされたインターバル内で、指定の信号エッジの数を合計します。

評価するエッジのタイプ（立ち上がり、立ち下がり、あるいはその両方）は、“Significant Edge”パラメータで指定します。

チャンネルにおけるデータ転送が有効になるのは、タスクが選択されている場合、または“IRQ”パラメータ（240 ページの「IRQ」を参照してください）が **Yes** に設定され、さらにその IRQ が実際に使用可能な場合だけです。

Use HW Trigger

デジタル入力信号の取り込みを外部トリガ信号と同調して行うかどうかを指定します。以下のモードを使用できます。

- **Yes**

  入力信号の取り込みは、外部トリガと同調して行われます。
- **No**

  入力信号の取り込みはポーリングモードで行われ、外部トリガ信号とは同調しません。

“Trigger Mode”パラメータは、ES1325-DIO サブシステムの“Globals”タブで、すべてのハードウェアチャンネルに共通な値として指定されます。

IRQ

入力信号の取り込み後に割込みが生成されるようにするかどうかを指定します。

- **Yes**

  IRQ ハンドラタスク（237 ページの「IRQ Handler Task」を参照してください）がアクティブ化され、データ取り込み後、データを ASCET モデルに転送するための割込み要求が生成されます。

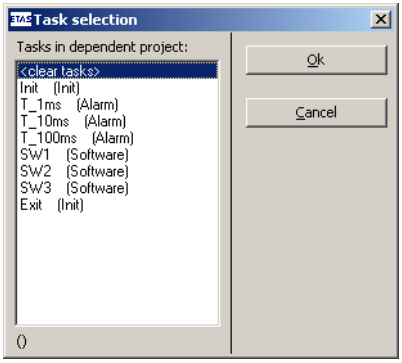
- **No**

  取り込まれた信号はバッファメモリに格納され、処理待ち状態となります。データ転送は、“Task”で指定された ASCET モデルのタスクにより、ポーリングモードで行われます。

“IRQ”オプションは、“Use HW Trigger”パラメータが **Yes** になっている場合に限り有効です（240 ページの「Use HW Trigger」を参照してください）。

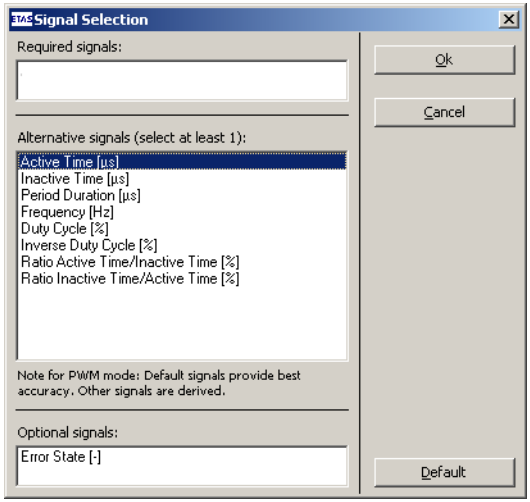
Activated Task

“Activated Task”フィールドをクリックすると、“Task Selection”というダイアログボックスが開きます。シグナルグループ受信時にアクティブ化されるタスクを選択します。



このオプションは、“IRQ”パラメータが **Yes** に設定されている場合のみ利用可能です（240 ページの「IRQ」を参照してください）。

有効なシグナルグループについては、このフィールドに [Select] と表示されるので、そこをクリックすると“Signal Selection”ダイアログボックスが開きます。



各シグナルごとにシグナルコンポーネントが割り当てられます。

“ChXXMsr”測定グループに使用できるシグナルは、以下のとおりです。

- State
- Active Time [µs]
- Inactive Time [µs]
- Period Duration [µs]
- Frequency [Hz]
- Duty Cycle [%]
- Ratio Active Time/Inactive Time [%]
- Ratio Inactive Time/Active Time [%]
- Additive Active Time
- Counter Value

オンラインの“ChXXAdj”調整グループ（192 ページの「“Globels”タブ」の「Adjustment Mode」を参照してください）については、使用のシグナルを以下できます。

- Time Delay [µs]
- Angle Delay [µs]
- Angle Interval [µs]

“Channel Mode”、“Use HW Trigger”、“HW Trigger Mode”パラメータの設定内容によっては、一部のシグナルをシグナルグループに必ず使用しなければならない場合や、デフォルト値が使用される場合があります。

Active State

入力信号のどちらのレベル（High または Low）をアクティブステートとするかを指定します。

- High

  入力信号の HIGH レベルが「アクティブ」ステートとして扱われます。
- Low

  入力信号の LOW レベルが「アクティブ」ステートとして扱われます。

Significant Edge

イベント（チャンネルモードが **`Event Counter`** の場合）または PWM 周期の開始（チャンネルモードが **`PWM Input`** の場合）を表わすエッジのタイプを選択します。

- Inactive-Active

  入力信号が非アクティブからアクティブに遷移する（243 ページの「Active State」を参照してください）と、イベントがトリガされます。
- Active-Inactive

  入力信号がアクティブから非アクティブに遷移する（243 ページの「Active State」を参照してください）と、イベントがトリガされます。
- Both

  入力信号が非アクティブからアクティブに遷移したときと、アクティブから非アクティブに遷移したとき（243 ページの「Active State」を参照してください）に、イベントがトリガされます。

この設定はチャンネルモードが **`Digital Input`** や **`Additive Time`** の場合には使用できません（239 ページの「Channel Mode」を参照してください）。

Hysteresis

入力チャンネルごとに、ヒステリシスのタイプを選択する必要があります。選択可能なオプションは以下のとおりです。

- TTL
- User defined

	TTL	User defined
下側のしきい値	1.728V	ユーザー定義
上側のしきい値	2.304V	ユーザー定義

User defined オプションを選択した場合には、ヒステリシスの下上のしきい値を定義する必要があります（244 ページの「Low Thresh. [V]」と 244 ページの「High Thresh. [V]」を参照してください）。

Low Thresh. [V]

各入力チャンネルに、入力信号の下側のしきい値を割り当てます。ここには、上側のしきい値より 1 ステップ以上小さい値を定義してください。

分解能		0.144V
値の範囲	最小	0V
	最大	上側のしきい値 − 0.144V

このオプションは、ヒステリシスモードとして **TTL** が選択されている場合は無効です（243 ページの「Hysteresis」を参照してください）。

High Thresh. [V]

各入力チャンネルに、入力信号の上側のしきい値を割り当てます。ここには、下側のしきい値より 1 ステップ以上大きい値を定義してください。

分解能		0.144 V
値の範囲	最小	下側のしきい値 ＋ 0.144V
	最大	36V

このオプションは、ヒステリシスモードとして **TTL** が選択されている場合は無効です（243 ページの「Hysteresis」を参照してください）。

Timeout [ms]

入力信号のステートが 1 回以上変化しなければならない時間を定義します。ここで定義された時間内に入力信号のステートが変化しなかった場合、所定のエラーステートが生成されます。

分解能		2.5ms
値の範囲	最小	0ms（この場合、タイムアウトは発生されません）
	最大	163837.5ms

10.14.4 Signals（ES1325-Input デバイス）

本項では、Input デバイスの各シグナル用オプションについて説明します。

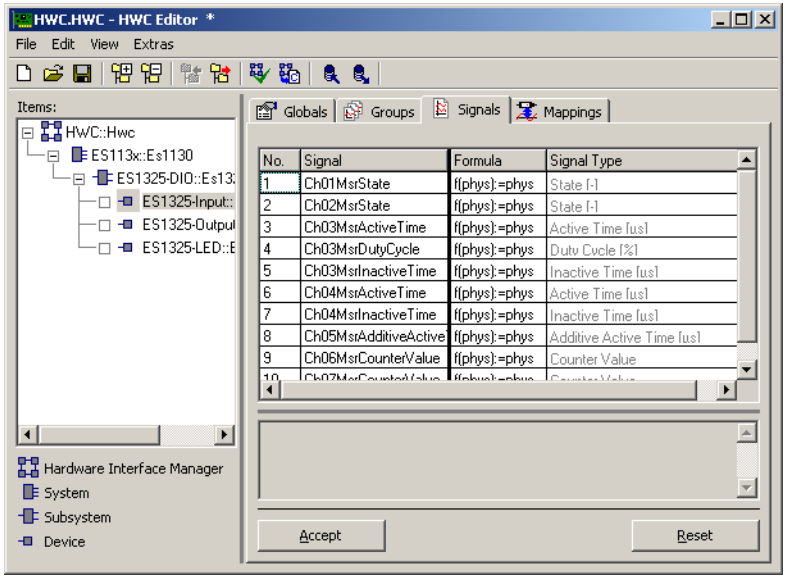


図 10-56 ES1325-DIO Input デバイスの“Signals”タブ

HW Channel

ES1325 の入力チャンネル（Ch01 ～ Ch16）が表示されます。

Signal Type

シグナルに割り当てられているシグナルタイプが表示されます。この割り当ては、“Groups”タブの“Signals”パラメータで変更できます。

10.14.5 Mappings（ES1325-Input デバイス）

このタブ上の設定は、すべてのデバイスについて共通です。125 ページの 7.4.5 項を参照してください。

10.14.6 Globals（ES1325-Output デバイス）

本項では、Output デバイスのグローバルオプションについて説明します。

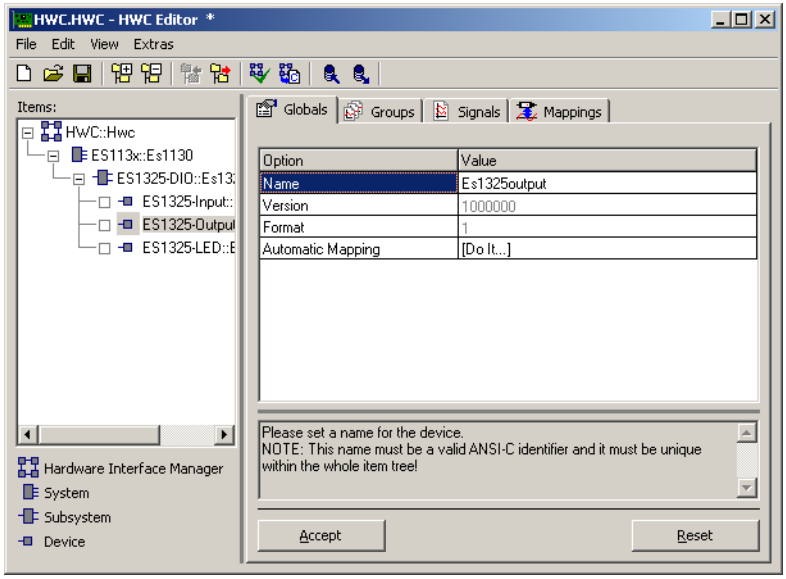


図 10-57 ES1325-DIO Output デバイスの“Globals”タブ

Automatic Mapping

このオプションで、シグナルを ASCET メッセージに自動的に割り当てます。

詳しくは 173 ページの「Automatic Mapping」を参照してください。

10.14.7 Groups（ES1325-Output デバイス）

本項では、Output デバイスの各シグナルグループ用オプションについて説明します。

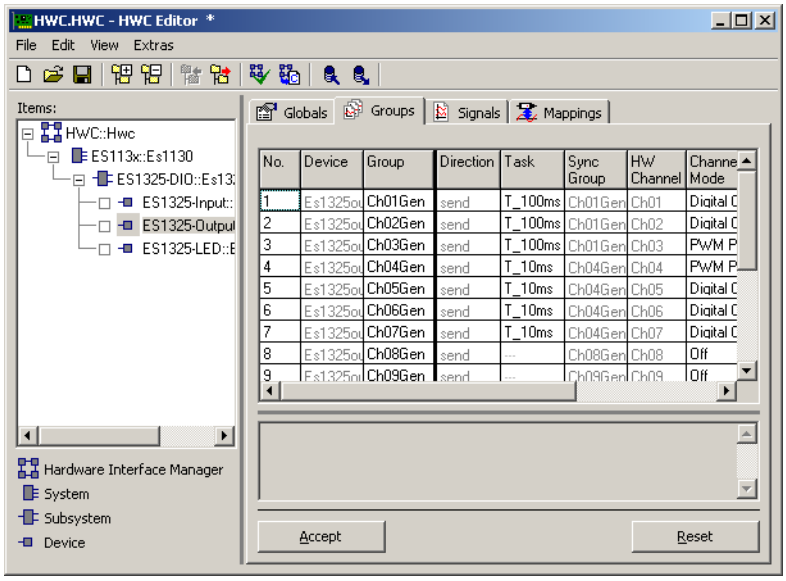


図 10-58 ES1325-DIO Output デバイスの“Groups“タブ

Sync Group

同じタスク（またはタスクグループ）によって同調して転送されるシグナルグループは、1 つのプロセスに割り当てられます。このプロセスには、この列に表示されているグループ名と同じ名前が付きます。

シグナルグループに複数のタスクが選択されている場合、“Task”フィールドの値が同じシグナルグループのプロセスのみ、1 つの共通プロセスに割り当てることができます。

このプロセスに割り当てられたシグナルグループは、同調して転送されます。

Hardware Channel

ES1325 の出力チャンネル（Ch01 ～ Ch16）

Channel Mode

ES1325 の各出力チャンネルについて、どのようなタイプの信号を出力するかを制定します。

- Off

  デジタル信号を出力しません。

- Digital Output

  要求に応じて、出力チャンネルのステートをアクティブまたは非アクティブに設定できます。この要求は、ミュレーションプロセッサからトリガできます。

- PWM Periodic Output

  出力時間の制限なしに PWM 信号が生成されます。

  生成される PWM 信号を定義するために、2 つのシグナルコンポーネントが必要です（250 ページの「Signals（ES1325-Output デバイス）」を参照してください）。

- PWM Interval Output

  PWM 機能により、単一パルスの生成や、所定インターバル分の PWM 信号の生成が行われます。最初の要求が処理されている間に受信された要求は、拒否されます。

チャンネルにおけるデータ転送が有効になるのは、タスクが選択されている場合、または“IRQ”パラメータ（240 ページの「IRQ」を参照してください）が **Yes** に設定され、さらにその IRQ が実際に使用可能な場合だけです。

Use HW Trigger

デジタル出力信号の出力を外部トリガ信号と同調して行うかどうかを指定します。以下のモードを使用できます。

- **Yes**

  信号出力は、外部トリガと同調して行われます。

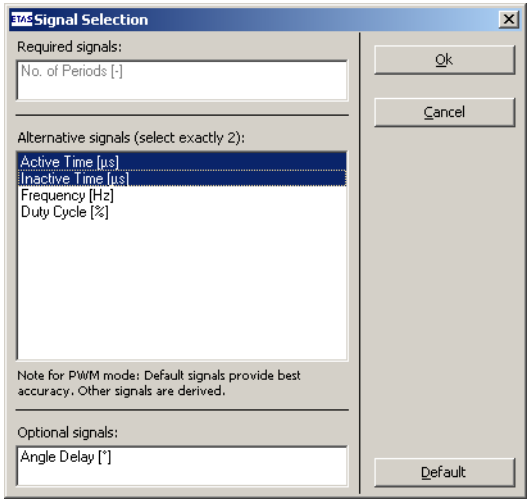
- **No**

  信号出力はポーリングモードで行われ、外部トリガ信号と同調しません。

“Trigger Mode”パラメータは、ES1325-DIO サブシステムの“Globals”タブで、すべてのハードウェアチャンネルに共通な値として指定されます。

Signals

有効なシグナルグループについては、このフィールドに [Select] と表示されるので、そこをクリックすると“Signal Selection”ダイアログボックスが開きます。ここで、生成されるシグナルを選択します。



各シグナルごとにシグナルコンポーネントが割り当てられます。

- State
- Active Time [µs]
- Inactive Time [µs]
- Frequency [Hz]
- Duty Cycle [%]
- Time Delay [µs]
- No. of Periods
- Angle Delay [°]

“Channel Mode”、“Use HW Trigger”、“HW Trigger Mode”パラメータの設定内容によっては、一部のシグナルをシグナルグループに必ず使用しなければならない場合や、デフォルト値が使用される場合があります。

Active State

出力信号のどちらのレベル（High または Low）をアクティブステートとするかを指定します。

- High

  出力信号の HIGH レベルが「アクティブ」ステートとして扱われます。

- Low

  出力信号の LOW レベルが「アクティブ」ステートとして扱われます。

10.14.8 Signals（ES1325-Output デバイス）

本項では、Output デバイスの各シグナル用オプションについて説明します。

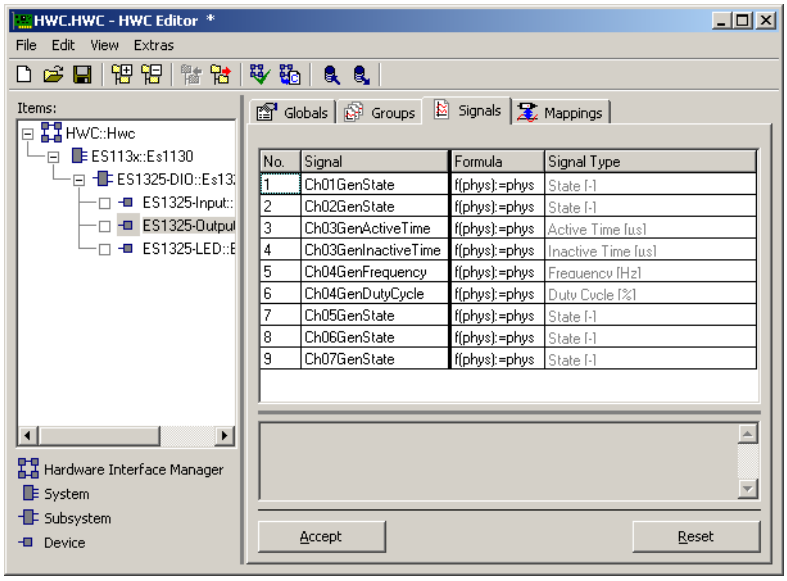


図 10-59 ES1325-DIO Output デバイスの“Signals”タブ

HW Channel

ES1325 の出力チャンネル（Ch01 ～ Ch16）

Signal Type

シグナルに割り当てられているシグナルタイプが表示されます。この割り当ては、“Groups”タブの“Signals”パラメータで変更できます。

10.14.9 Mappings（ES1325-Output デバイス）

このタブ上の設定は、すべてのデバイスについて共通です。125 ページの 7.4.5 項を参照してください。

10.14.10 Globals（ES1325-LED デバイス）

本項では、LED デバイスのグローバルオプションについて説明します。

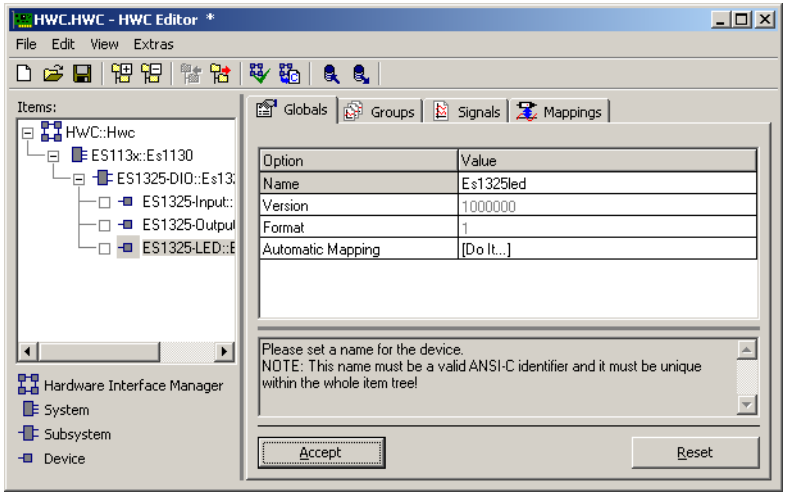


図 10-60 ES1325-DIO LED デバイスの“Globals”タブ

Automatic Mapping

このオプションで、シグナルを ASCET メッセージに自動的に割り当てます。

詳しくは 173 ページの「Automatic Mapping」を参照してください。

10.14.11 Groups（ES1325-LED デバイス）

本項では、LED デバイスの各シグナルグループ用オプションについて説明します。

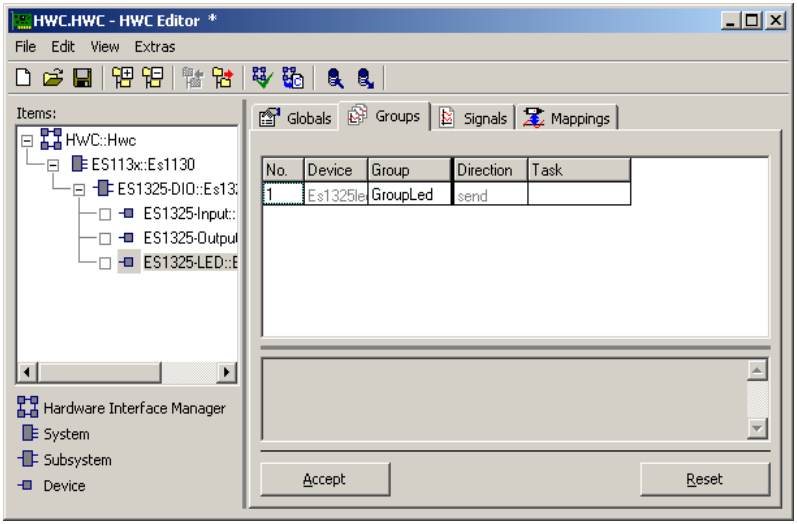


図 10-61 ES1325-DIO LED デバイスの“Groups”タブ

ES1325-LED デバイスの“Groups”タブには、このデバイスに固有なオプションはありません。一般的なオプションの設定については 123 ページの 7.4.3 項に説明されています。

Task

デバイスを使用するタスクをこのフィールドで選択します。

10.14.12 Signals（ES1325-LED デバイス）

本項では、LED デバイスの各シグナル用オプションについて説明します。

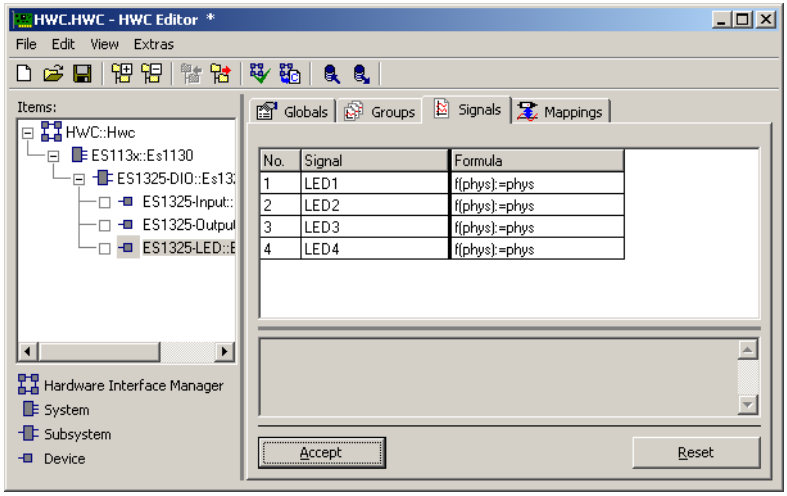


図 10-62 ES1325-DIO LED デバイスの“Signals”タブ

ES1325-LED デバイスの“Signals”タブには、アイテム固有の列は定義されていません。このアイテムに可能な設定については 124 ページの 7.4.4 項に説明されています。

10.14.13 Mappings（ES1325-LED デバイス）

このタブ上の設定は、すべてのデバイスについて共通です。125 ページの 7.4.5 項を参照してください。

10.15 ES1330-PWM

ES1330 は、PWM 信号の取り込みと出力を行うためのカウンタボードです。このボードには 6 個のカウンタコンポーネント（Am9513A）が搭載されていて、各カウンタコンポーネントにはカウント周波数が 4MHz の 5 つの 16 ビット幅カウンタがあります。30 点のカウンタ入力と 16 点のカウンタ出力が ES1330 の 6 点の内部ポート（ポート 1 ～ポート 6）に分けられ、各カウンタコンポーネントがこれらのポートを 1 つずつ占有します。ポートへのアクセスは、ES1330 の D-

SUB コネクタを通じて行われます。HWC エディタのアイテムリストでは、ES1330 カウンタボードをサブシステムとして追加し、各カウンタコンポーネントをデバイスとして追加します。

注記

ES1330 ボードのユーザーズガイド（`EtasManuals¥ASCET5.2¥ES1000` ディレクトリに格納されています）の「Connector X1: Diginal Inputs and Outputs」の項に、使用できるカウンタ入出力ポートの一覧が記載されています。

10.15.1 Globals（ES1330-PWM サブシステム）

本項では、ES1330-PWM サブシステムのグローバルオプションについて説明します。

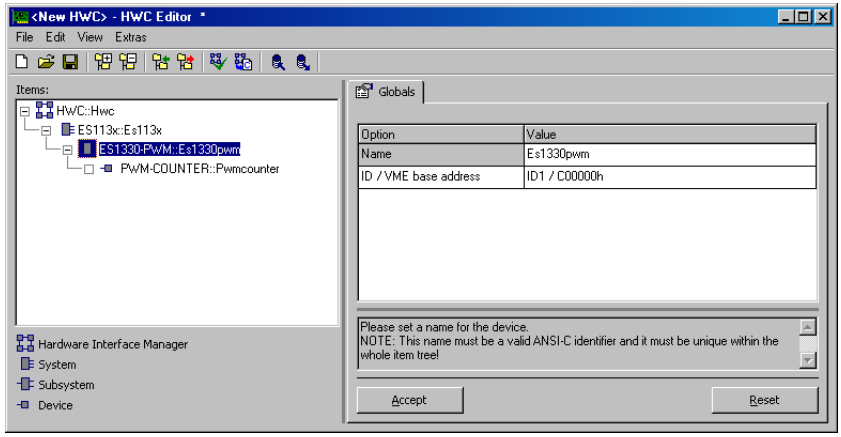


図 10-63 ES1330-PWM サブシステムの“Globals”タブ

ID / VME base address

この行では、VME ベースアドレスの設定を行います。この設定はボード上のジャンパ設定と一致している必要があります。ES1330 の場合、4 種類の異なる VME ベースアドレス（`ID1/C00000h`、`ID2/C00100h`、`ID3/C00200h`、`ID4/C00300h`）から選択することができます。つまり、1 つの ES1000 システムで最大 4 枚の ES1330 を稼働させることができ、1 枚の ES1330 はベースアドレスから始まる 256 ワードのアドレススペースを占有します。

10.15.2 Globals（PWM-COUNTER デバイス）

本項では、PWM-COUNTER デバイスのグローバルオプションについて説明します。

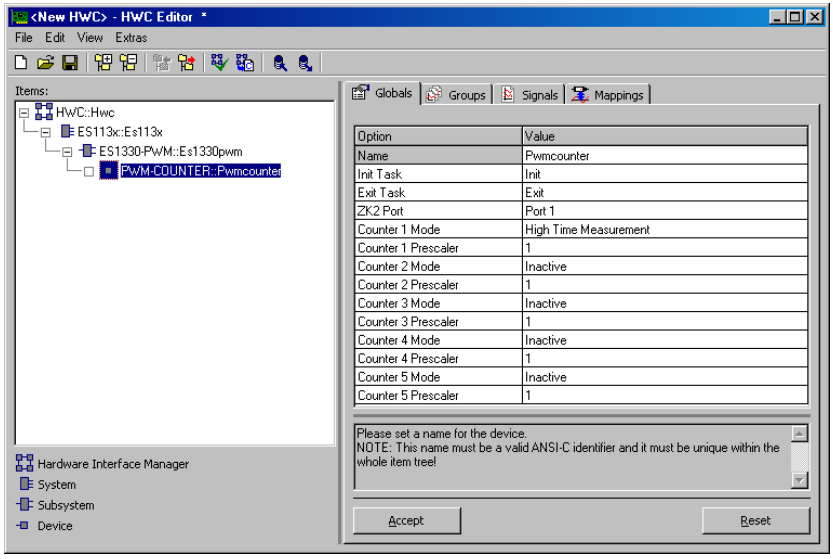


図 10-64 PWM-COUNTER デバイスの“Globals”タブ

Init Task

ES1330 を初期化するためのタスクを指定します（タイプ：Init、アプリケーションモード：active）。

Exit Task

実験を終了するときに ES1330 の終了処理を行なうタスクを指定します（タイプ：Init、アプリケーションモード：inactive）。

ZK2 Port

ポート、つまりカウンタコンポーネントを指定します。各カウンタコンポーネントごとに専用のポートが使用されます。

Counter 1 Mode ～ Counter 5 Mode

ここでは、カウンタコンポーネントの 5 個のカウンタのそれぞれについて、カウンタモードを選択します。選択できるカウンタモードは以下のとおりです。

- `inactive`

  カウンタは使用しません。

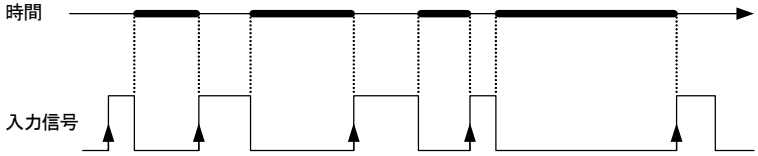
- `PWM-Generator (PWM_Gen)`

  カウンタを使用して PWM 信号を生成します。
- `Period-Measurement (P_Meas)`

  カウンタを使用して PWM 信号の周期を測定します。
- `High Time Measurement`

  カウンタを使用して、PWM 信号（アクティブ High）のアクティブパルス幅を測定します。
- `Low Time Measurement`

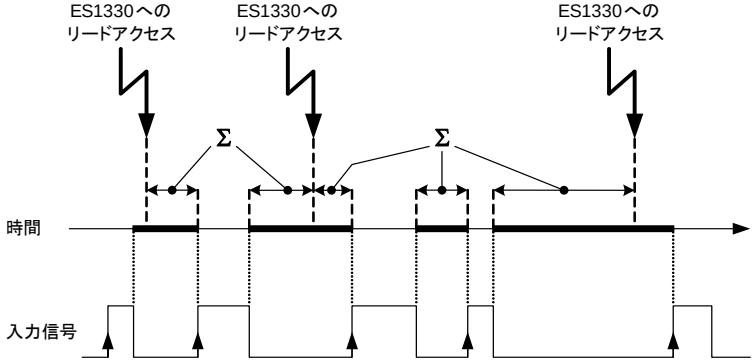
  カウンタを使用して、PWM 信号（アクティブ Low）のアクティブパルス幅を測定します。



- `Additive High Time Measurement`

  シミュレーションプロセッサは、通常、ES1330 から一定の周期で測定値を取り込みますが、この「High タイムの合計」モードにおいては、2 回の連続するリードアクセス間に PWM 信号（アクティブ High）がアクティブであった時間をすべて合計します。
- `Additive Low Time Measurement`

  シミュレーションプロセッサは、通常、ES1330 から一定の周期で測定値を取り込みますが、この「Low タイムの合計」モードにおいては、2 回の連続するリードアクセス間に PWM 信号（アクティブ Low）がアクティブであった時間をすべて合計します。



Counter 1 Prescaler ～ Counter 5 Prescaler

これらのパラメータは、4MHz の入力周波数用のプリスケーラを指定するためのものです。スケーリング後の入力周波数は、最初の 4 つのカウンタ用のクロック信号として使用されます。

プリスケーラを使えば、周期の長い PWM 信号を生成したり評価することができますが、PWM 信号の分解能は低下します。プリスケーラは、1、10、100、1000 および 10000 から選択できます。

10.15.3 Groups（PWM-COUNTER デバイス）

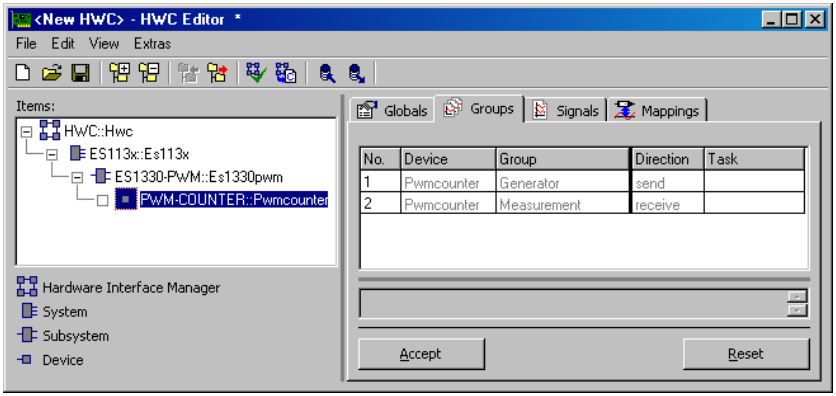


図 10-65 PWM-COUNTER デバイスの“Groups”タブ

ES1330-PWM サブシステムの“Groups”タブには、このサブシステム固有の項目はありません。

10.15.4 Signals（PWM-COUNTER デバイス）

本項では、PWM-COUNTER デバイスの各シグナルごとのオプションについて説明します。

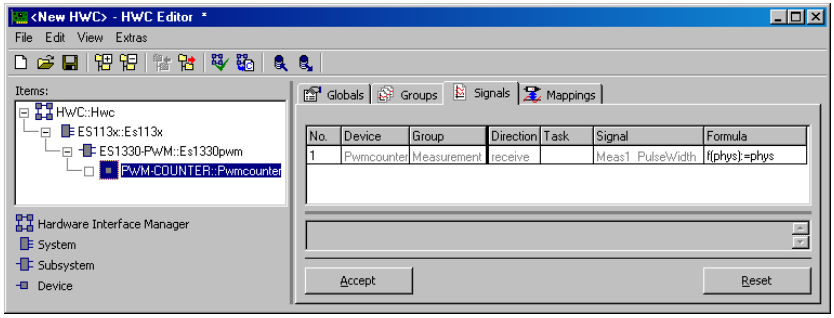


図 10-66 PWM-COUNTER デバイスの“Signals”タブ

No.

シグナルの番号と意味は、“Globals”タブの“ZK2 Port”および“Counter X Mode”設定により決まります。

Signal

シグナルの名前（ANSI-C）を具体的に定義します。以下の命名規則が適用されます（x =カウンタの番号）。

- `Genx_Frequency`

  PWM 出力チャンネルの周波数（Hz）
- `Genx_DutyCycle`

  PWM 出力チャンネルのデューティサイクル（%）
- `Measx_Period`

  PWM 評価チャンネルの周期（秒）
- `Measx_DutyCycle`

  PWM 信号のパルス幅測定（単一パルス幅、またはアクティブタイムの合計）の結果

10.15.5 Mappings（PWM-COUNTER デバイス）

このタブ上の設定は、すべてのデバイスについて共通です。125 ページの 7.4.5 を参照してください。

11 チュートリアル

この章では、サンプルプロジェクトを使用して、INTECRIO で実験を行う方法（11.1 項）や、各種ボードの設定方法（ES1222 - 11.2 項、ES1303 - 11.3 項、ES1325 -11.4 項と 11.5 項 ）について学習します。

ASCET-RP V5.5 には 2 つのサンプルファイル（`INTECRIO_Tutorial.exp` および `RRIOTutorial.exp`）が含まれていて、製品をインストールすると、ASCET のインストールディレクトリのサブディレクトリ（`Ascet5.2¥export`）に、このファイルが格納されます。

サンプルファイルは、新しいデータベース、または既存のデータベースにインポートして使用します。

注記

このチュートリアルは、ETAS ネットワークマネージャを使用しないことを前提にした操作内容となっています。そのため、ETAS ネットワークマネージャを使用してチュートリアルを実行すると、システムの挙動が若干異なります。詳しくは、2.2 項と第 4 章を参照してください。

新しいデータベースを作成する：

- ASCET のコンポーネントマネージャで、**File → New Database** メニューコマンドを選択します。

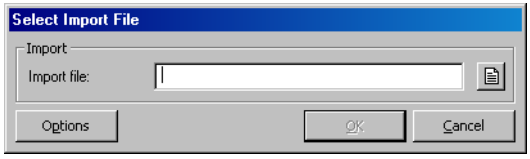
  “New Database”ダイアログボックスが開きます。
- 新しいデータベースの名前を入力します。
- OK をクリックします。

  新しく作成されたデータベースが開きます。

サンプルファイルをインポートする：

- コンポーネントマネージャで **File → Import** を選択します。

  “Select Import File”ダイアログボックスが開きます。

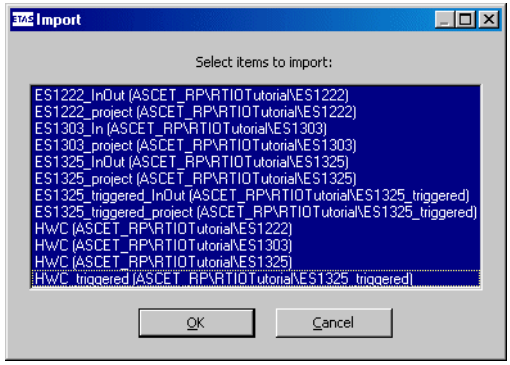


-  ボタンをクリックし、ASCET のインストールディレクトリの `ASCET5.2¥export` サブディレクトリから、`RTIOTutorial.exp` というファイルを開きます。

  OK ボタンが有効になります。

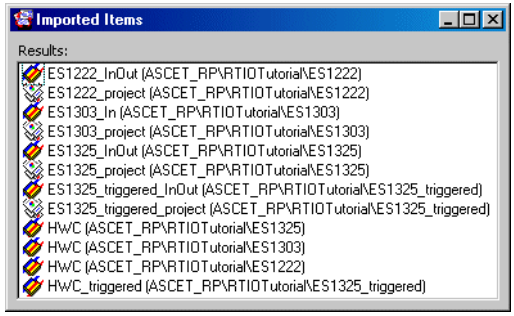
  ここではデフォルトのインポートオプションを使用するので、オプション設定を変更する必要はありません。
- **OK** をクリックします。

  “Import” ダイアログボックスが開きます。
- すべてのコンポーネントが選択された状態になっていることを確認してください。



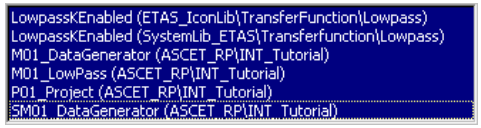
- **OK** をクリックします。

  ファイルがデータベースにインポートされます。インポートされたデータベースアイテムが “Imported Items” ダイアログボックスに表示されます。



- “Imported Items”ダイアログボックス を閉じます。
- 同じ方法で `INTECRIO_Tutorial.exp` というファイルをインポートします。

  インポートの際は、すべてのコンポーネントが選択されていることを確認してください。



ASCET 用 PC に、社内 LAN 用のネットワークカードが別にインストールされている場合、2 枚のカードの IP アドレスの衝突を防ぐため、以下のようなネットワーク設定を行う必要があります。

注記

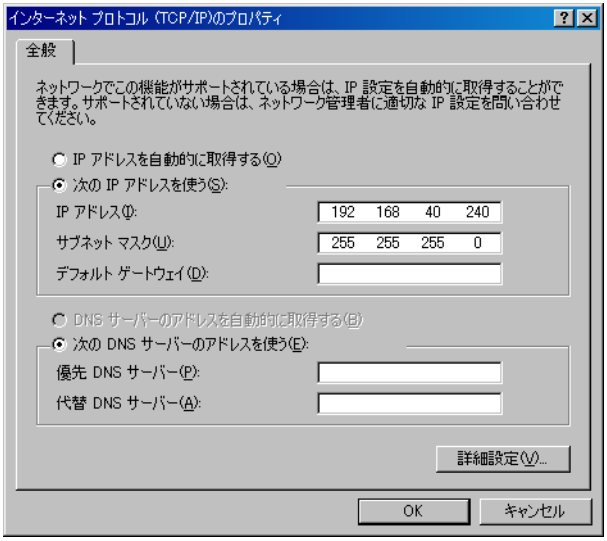
ETAS ネットワークマネージャを使用している場合、以下の操作は不要です。

TCP/IP プロトコルオプションを設定する：

- DHCP サービスを無効にします。
- IP アドレスフィールドに `192.168.40.240` と入力します。

- サブネットマスクフィールドに 255.255.255.0 と入力します。

（Windows® 2000 の場合）



- DNS サービスを利用する場合は、社内 LAN のローカル設定を使用してください。
- WINS サービスを無効にします。
- IP 転送オプションが無効になっていることを確認します。

11.1 チュートリアル - INTECRIO を使用した実験

この項では、ASCET のサンプルプロジェクトを INTECRIO に転送して INTECRIO で実験を行い、さらに ASCET 上で INTECRIO のバックアニメーション機能（60 ページ参照）を利用します。ここでは ASCET のプロジェクトや INTECRIO のワークスペースを作成する作業は行いません。必要なファイルはすべて用意されています。

- エクスポートファイル `INTECRIO_Tutorial.exp` には、ASCET プロジェクトとその他必要なコンポーネントがすべて含まれています。

  ASCET プロジェクト `P01_Project` には、ステートマシン（`SM01_DataGenerator`）で記述されたデータジェネレータ（`M01_DataGenerator`）が含まれています。`PMode` というパラメータを変更し、データジェネレータがのこぎり歯を生成するようにするか（`PMode=1`）、または三角波を生成するようにするか（`PMode=2`）を指定します。生成されたデータは、同じプロジェクトに含まれるローパスフィルタ（`M01_LowPass` モジュール）の入力シグナルとなります。

- INTECRIO_Tutorial_Workspace というフォルダには、3 つの INTECRIO ワークスペース（SystemProject_ES1130、SystemProject_ES1135、SystemProject_ES910）が含まれているので、実際のハードウェアに対応するものを使用してください。

11.1.1 準備

まず始めに、実験の準備を行います。

ASCET のインストール用 CD の ETAS¥ASCET5.2¥export というフォルダに格納されている INTECRIO のワークスペースを使用します。

INTECRIO のワークスペースをコピーする：

- INTECRIO_Tutorial_Workspace ディレクトリを PC のハードディスク（以下のようなパス）にコピーします。

  ETASData¥INTECRIO2.1¥INTECRIO_Tutorial_Workspace

ターゲットを選択する：

- プロジェクト P01_Project をプロジェクトエディタで開きます。
- “Project Properties”ウィンドウを開きます。
- “Build”ノードで、ターゲットに Prototyping、コンパイラに GNU-C V3.4.4 (PowerPC) を選択します。

  このターゲット用にオペレーティングシステムのコンフィギュレーションが設定されていない場合、**Operating System → Copy from Target** で設定をコピーします。

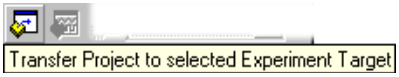
11.1.2 プロジェクトの転送

次にプロジェクトを INTECRIO に転送します。

プロジェクトを INTECRIO に転送する：

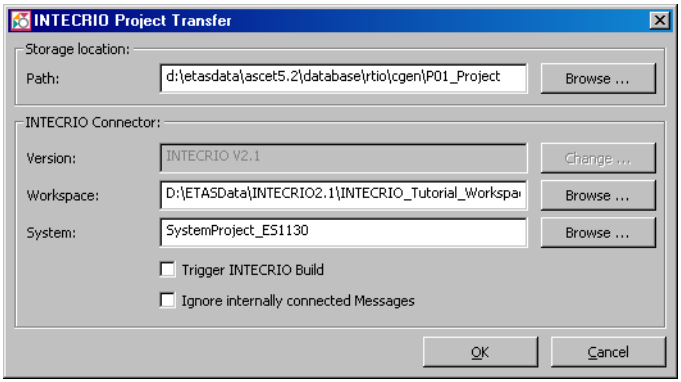
- プロジェクト P01_Project を開きます。

  “Experiment Target”コンボボックスには INTECRIO が選択されています。また **Transfer Project to selected Experiment Target** ボタンと **Reconnect to Experiment of selected Experiment Target** ボタンが有効になっています。



- **Transfer Project to selected Experiment Target** ボタンをクリックします。

  “INTECRIO Project Transfer” ダイアログボックスが開きます。



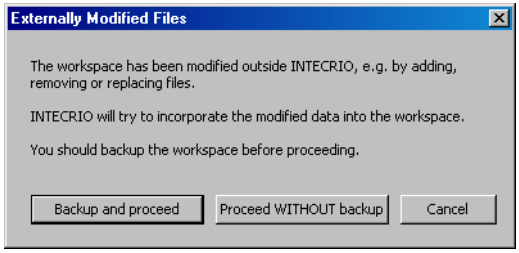
- “Path” フィールドに、生成されたファイルを格納するパスを入力します。
- “Version” フィールドの隣の **Change** ボタンをクリックして、使用する INTECRIO のバージョンを選択します。

  PC 上に 1 つのバージョンの INTECRIO しかインストールされていない場合、このボタンは無効になります。
- “Workspace” フィールドの隣の **Browse** ボタンをクリックして、使用するワークスペースを指定します。
- “System” フィールドの隣の **Browse** ボタンをクリックして、使用するハードウェアに適したシステムプロジェクトを指定します。

  INTECRIO が起動していない場合は、ここで起動されます。
- **OK** をクリックします。

  INTECRIO での作業に必要なすべてのファイルが生成され、指定されたディレクトリに保存されます。そして ASCET プロジェクトが INTECRIO にインポートされて `P01_Project` という名前のモジュールとして保存され、選択されたプロジェクトに自動的に追加されます。

ワークスペースをコピーすると、以下のようなメッセージが表示される場合があります。



この場合、**Proceed without Backup** で処理を続行してください。

11.1.3 INTECRIO での実験

次に INTECRIO でオペレーティングシステムを設定し、ビルド処理を開始して、INTECRIO での実験を行います。

INTECRIO のオペレーティングシステムを設定する：

- INTECRIO ウィンドウに切り替えます。
- `Systems` フォルダで、使用するシステムプロジェクトを選択し、ショートカットメニューから **Set As Active Project** を選択します。
- **System → OS Configuration** を選択します。

  OSC オペレーティングシステムエディタが開きます。サンプルプロジェクトの構成は非常に単純なので、自動設定を行えます。
- **System → OS Auto mapping** を選択します。

  `UserAppMode` というアプリケーションモードの `auto_10msTask` というタスクが生成されます。ASCET プロジェクトの 2 つのプロセスがこのタスクに割り当てられます。

  これ以外の設定を行う必要はありません。

INTECRIO のビルド処理を開始する：

- INTECRIO ウィンドウから **Integration → Build** を選択します。

または

- 2 回目以降のビルド処理を行う場合は、**Integration → Rebuild** を選択します。

  ビルド処理が開始されます。INTECRIO ウィンドウの最下部の“Log Window”というボックスに、実行状況が表示されます。

  ビルド処理が正常に終了すると、最終行に以下のメッセージが表示されます。

  ```
  Action succeeded
  ```

  ```
  The active system project has been
  set into the „Build" mode
  ```

INTECRIO の実験を開始する：

1. 実験環境を開く

   - INTECRIO ウィンドウで、**Experiment → Open Experiment** を選択します。

     INTECRIO の実験環境が個別のウィンドウとして開き、実験がロードされます。

2. 実験を開始する

   - INTECRIO の実験環境で、**Experiment → Download** を選択します。

     実効形式のファイル（プロトタイプ）がハードウェアにロードされます。

   - **Experiment → Start OS** を選択します。

     シミュレーションが開始されます。

   - **Experiment → Start Experiment** を選択します。

     測定が開始されます。

バックアニメーション機能を使用する際は、INTECRIO の実験環境で測定ウィンドウや適合ウィンドウを開く必要がありません。しかし ASCET でのバックアニメーションにおいてはオシロスコープが使用できないため、INTECRIO の実験環境にはオシロスコープ（`Oscilloscope1.osc`）が用意されていて、その他にも 2 つの適合インストゥルメントが用意されています。これらのインストゥルメントが自動的に開かない場合、以下のようにして、定義済みの実験をマニュアル操作で開いてください。

実験環境をロードする：

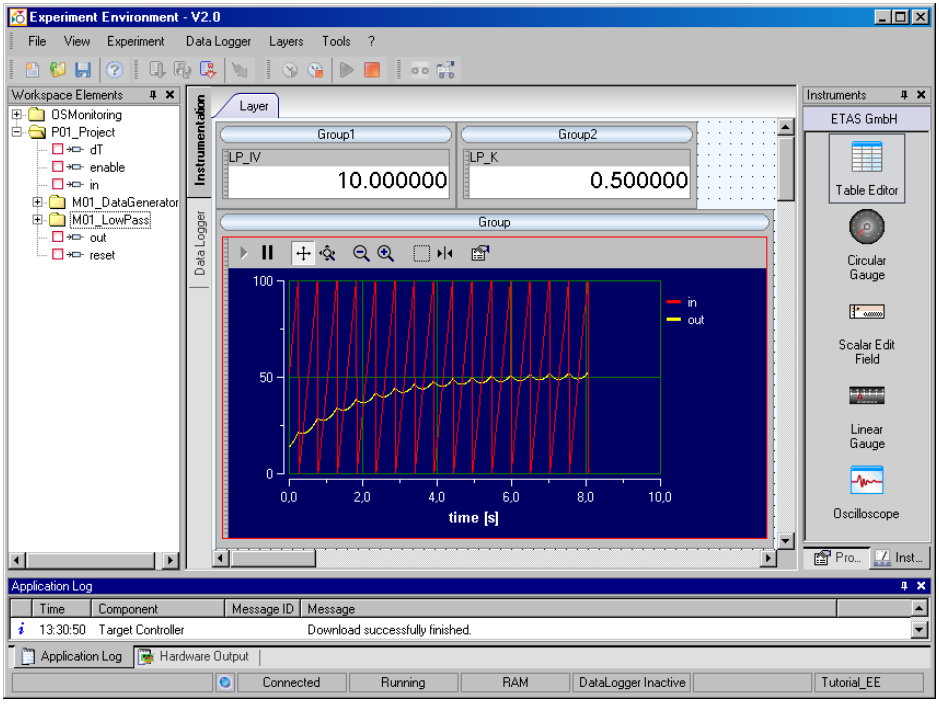
- INTECRIO の実験環境で、**File → Open Experiment** を選択します。
- 確認メッセージを確認します。

  ファイル選択ダイアログボックスが開きます。

- INTECRIO ワークスペースのサブディレクトリ `EE¥Experiments¥INTECRIO_Tutorial` サブディレクトリから、`INTECRIO_Tutorial.eex` ファイルを選択します。

  シミュレーションはすでに開始されているので、すぐに値が表示されます。

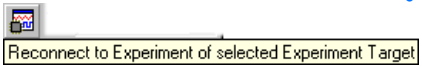
INTECRIO の実験環境は以下のようになります。



11.1.4 バックアニメーションの使用

ASCET からバックアニメーションを開始します。この際、INTECRIO で実験が実行されている必要があります。

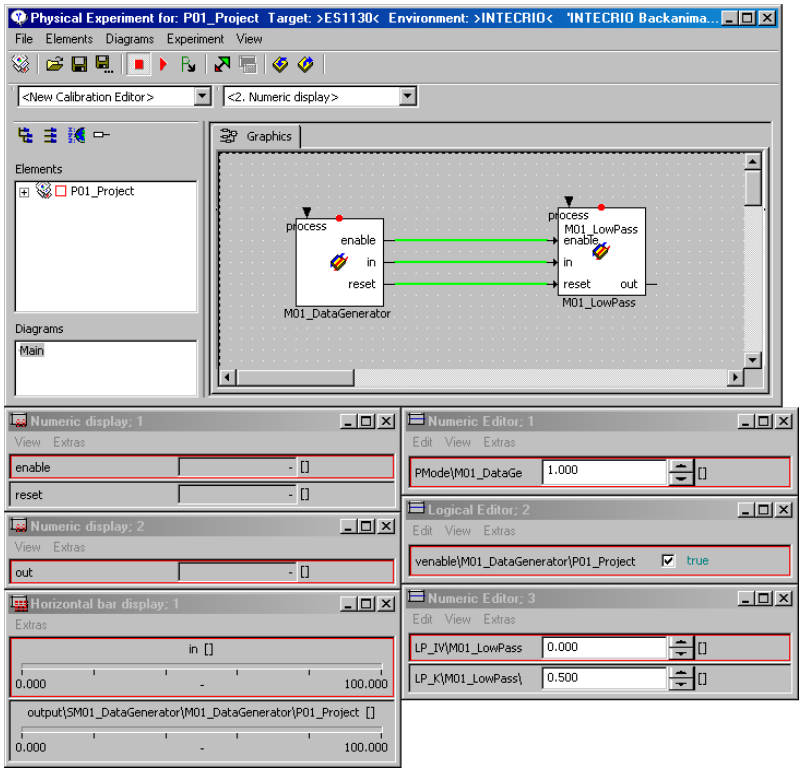
バックアニメーションを行う：

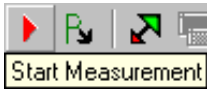


- ASCET プロジェクトエディタで **Reconnect to Experiment of Selected Experiment Target** ボタンをクリックします。

  INTECRIO の実験への接続が確立され、“Physical Experiment...” ウィンドウが開きます。

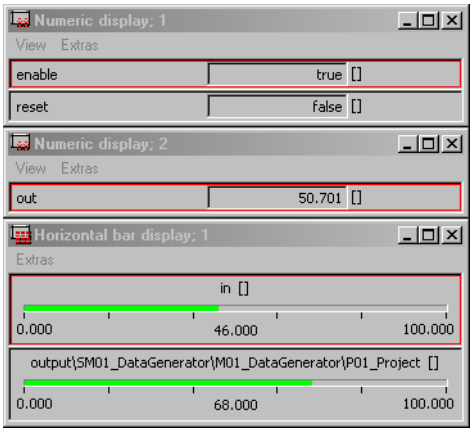
- “Environment Browser”ウィンドウで、INTECRIO を実験環境として選択します。





- **Start Measurement** ボタンをクリックします。

  ASCET 環境内の測定が開始され、測定ウィンドウに測定値が表示されます。



適合作業は、ASCET と INTECRIO のいずれの実験環境でも行えます。変更された値は INTECRIO に転送され、そこに表示されます。

値の適合を行う：

- ASCET で“`Numeric Editor; 3`”ウィンドウ内の変数 `LP_IV` の値を変更します。

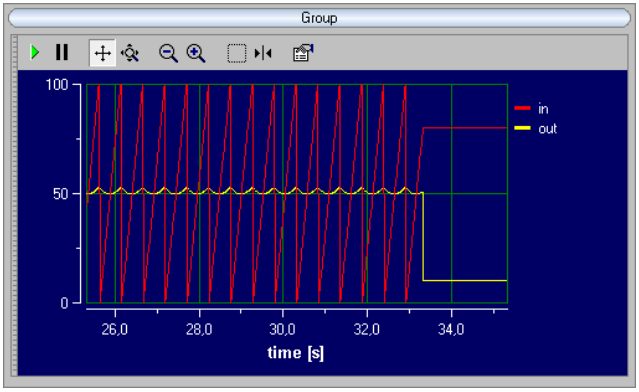
  INTECRIO の実験環境上で、左側の適合インストゥルメントの値が更新されます。

- INTECRIO の実験環境上で、“Group1”ウィンドウ内の変数 `LP_IV` の値を変更します。

  ASCET の実験環境上で、“`Numeric Editor; 3`”ウィンドウ内の値が更新されます。

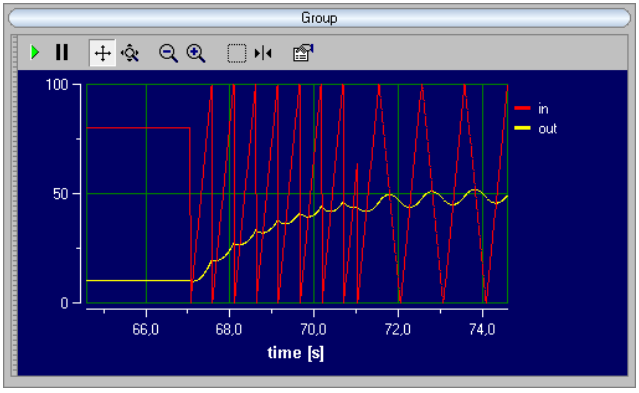
- ASCET で“Logical Editor; 2”ウィンドウ内のパラメータ venable の値を変更します。

  シグナルジェネレータの値がその時点の値のまま固定され、ローパスフィルタは初期値 LP_IV に設定されます。



- パラメータの値をtrueに戻し、PModeを2に設定して別のシグナルジェネレータを選択します。

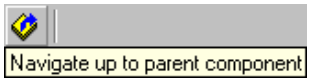
  INTECRIO のオシロスコープの表示は以下のように変わります。



ASCET コンポーネントを表示する：

- ASCET の“Physical Experiment ...”ウィンドウの“Graphics”タブ上のコンポーネントをダブルクリックして、内容を表示します。

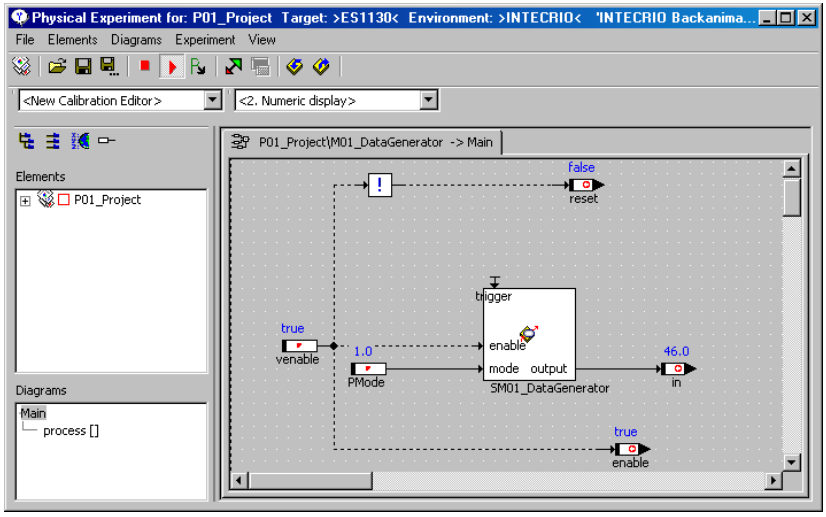
  “Physical Experiment ...”ウィンドウにそのコンポーネントの内容が表示されます。



  このようにしてモデルの全階層をナビゲートすることができます。1 つ上の階層に移動するには、**Navigate up to parent component** ボタンをクリックするか、または空白部分をダブルクリックします。

- **View → Monitor All** を選択します。

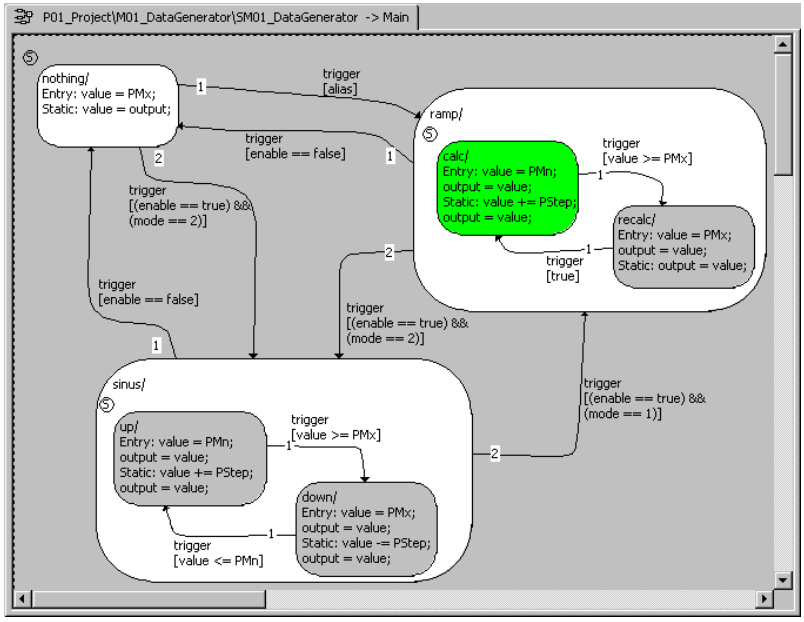
  各エレメントの上部に、その現在値が表示されます。



- モデル内をナビゲートし、ステートマシン `SM01_DataGenerator` まで移動します。

- いずれかのステートを右クリックしてショートカットメニューを開き、`Animate States` を選択します。

  ステートダイアグラム内で、現在のステートがカラー表示されます。



11.2 チュートリアル - ES1222（CAN-IO）

インポートされた ES1222 用モデルには、CAN チャンネル 1 から送信されて CAN チャンネル 2 で受信される 3 つの変数が含まれています。296 ページの図 11-3 のように**両チャンネルをターミネータ付きケーブルで接続すると、**オンライン実験においてこれらの変数の送受信が行われ、その状況を実験環境ウィンドウでモニタすることができます。

モデルの内容はすべて完成していて、ES1222 を RTIO に統合するための条件（タスク、メッセージ、HWC モジュール）が揃っています。あとは、HWC エディタでハードウェアコンフィギュレーションを設定し、コード生成を行なうだけです。

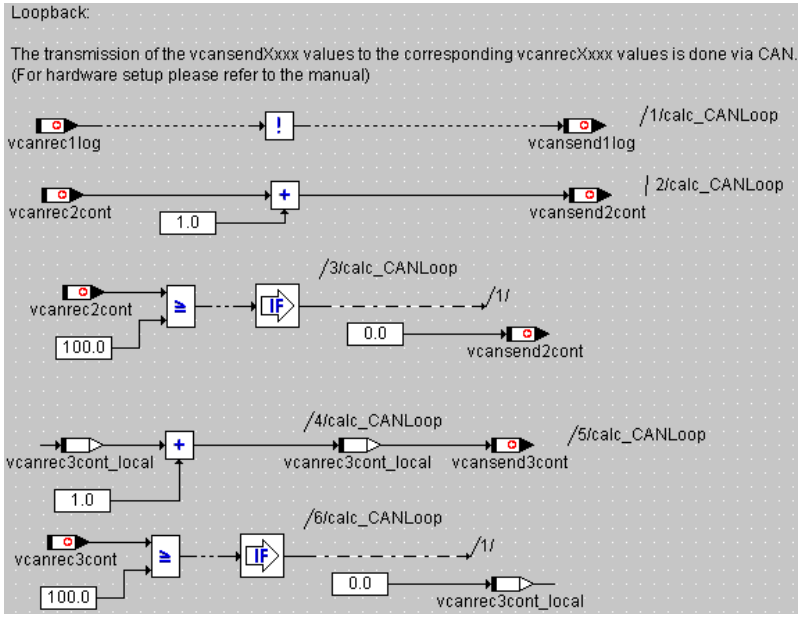


図 11-1 ES1222 モデルの ES1222_InOut モジュール

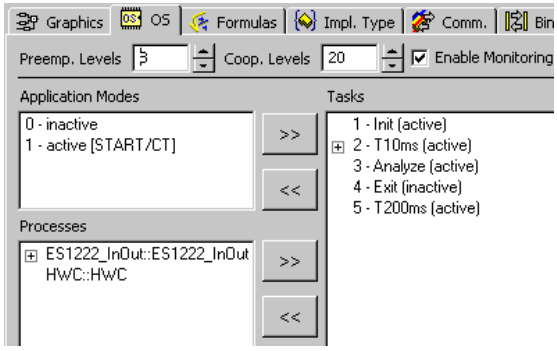
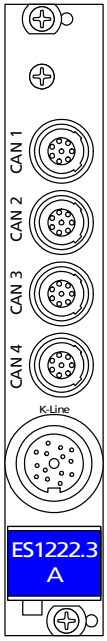


図 11-2 ES1222 モデルの OS エディタ設定

11.2.1 ES1222 ボード

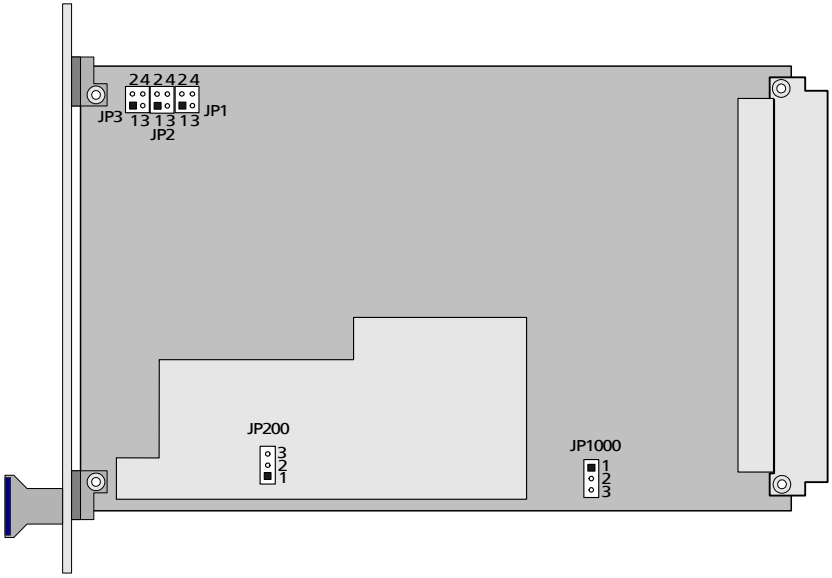
以下の図は、ES1222CAN ボードのフロントパネルの概観です。



4 つのインターフェースポート（CAN1 ～ CAN4）はそれぞれ独立していて、電気的に絶縁されています。また K ワイヤポートは、外部デバイスを VME バスシステムに接続するためのシリアル K ／ L ラインインターフェース用のものです。

ES1222 のジャンパ設定

実験を行うためには、ボード上の各ジャンパスイッチを以下の表のように正しく設定してください。



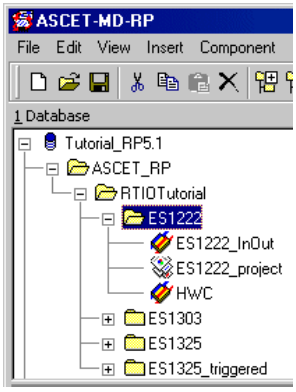
ジャンパスイッチ	ピンの設定	意味
JP1	オープン	CAN1 と CAN2 をループバック接続します。
JP2	オープン	
JP3	オープン	
JP1000	1-2 をクローズ	
JP200 (ES1222.3 のみ)	(Don't Care)	K Line はここでは使用しません。

複数の ES1222 ボードの使用

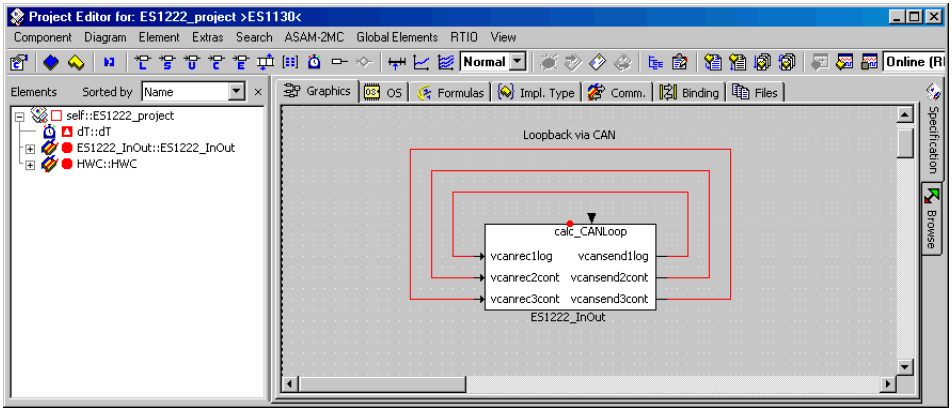
1 台の ES1000 システム内で、最大 4 枚の ES1222 ボードを使用できます。

サンプルプロジェクトを開く：

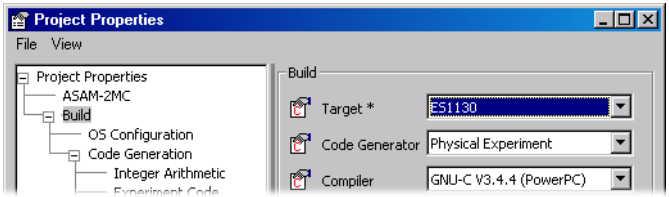
- コンポーネントマネージャで ASCET_RP/RTIOTutorial/ES1222 というフォルダを選択します。



- ES1222_project というプロジェクトを選択します。
- プロジェクトを開きます。



- **Project Properties** ボタンをクリックします。
  “Project Properties”ウィンドウが開きます。



- “Build”ノードで以下のオプションを設定します。
  `Target: >ES1130< または >ES1135<`
  `Compiler: GNU-C * (PowerPC)`

注記

RTIO との通信に使用できるメッセージは、“Exported”として宣言されたメッセージのみです。

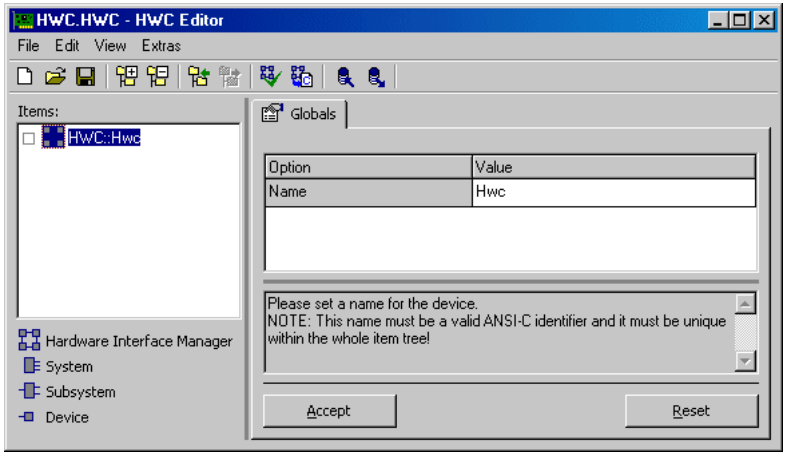
注記

通常、ハードウェアコンフィギュレーションを作成するには、まず HWC という C コードモジュールを作成し、それをプロジェクトに組み込んでからその内容を編集しますが、このチュートリアルでは、すでにこの HWC モジュールがプロジェクトに組み込まれています。

HWC エディタを開く：

- プロジェクトエディタで **RTIO → Open Editor** を選択します。

  HWC エディタが開きます。



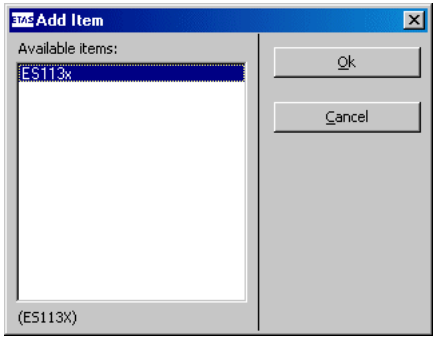
ハードウェアコンフィギュレーション（HWC）を作成する：

各ハードウェアは、アイテムリスト内にツリー構造の形で定義します。このツリーのルートには、必ず HWC というアイテムが存在します。

- HWC エディタで **Edit → Add Item...** を選択します。

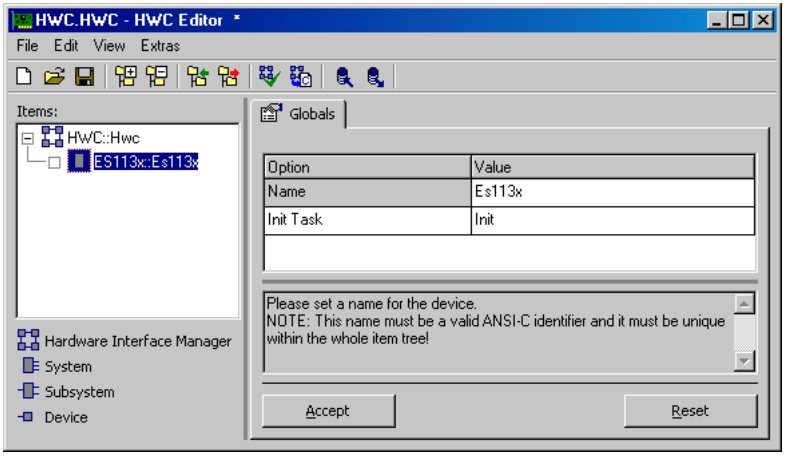
または

- **Add Item...** ボタンをクリックします。
  “Add Item”ダイアログボックスが開きます。



現在選択されているアイテムの次の階層レベルに追加できるアイテムの一覧が表示されます。

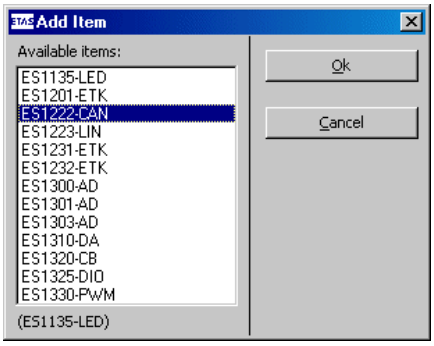
- ES113x を選択します。
  このアイテムは、ES1000.x システムで使用される PowerPC プロセッサボード ES1130 または ES1135 に相当します。
- **OK** をクリックします。
  “Items”リストに ES113x が追加されます。



- “Items”リストから ES113x を選択します。

“Globals”タブには、“Init Task”オプションの欄に `Init` というデフォルトのタスク名が表示されます。サンプルプロジェクトの OS エディタで同じ名前の Init タスクが定義されているので、ここで他のタスクを選択する必要はありません。

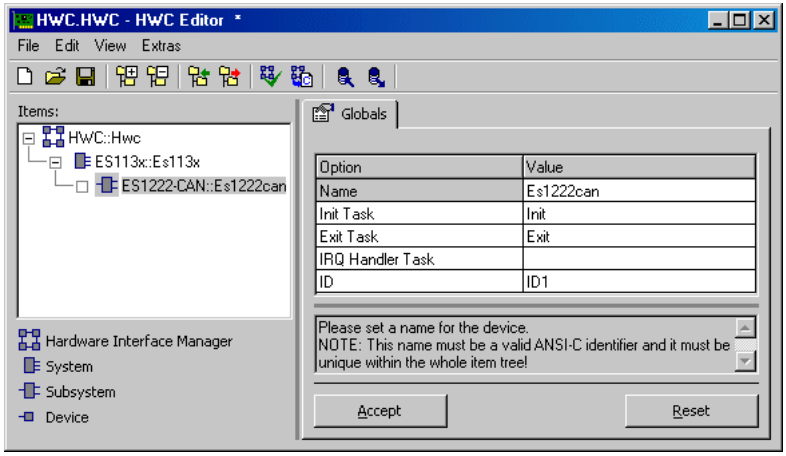
- **Edit → Add Item...** を選択して、次の階層レベルで選択可能なアイテムの一覧を開きます。



- ES1222-CAN というアイテムを選択します。

  このアイテムは、ES1222 CAN + K-Line インターフェースに相当します。
- **OK** をクリックします。

  “Items” リストに ES1222-CAN というアイテムが追加されます。

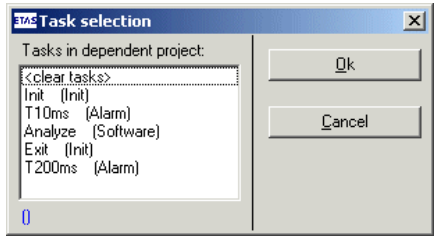


- “Items” リストから ES1222-CAN を選択します。

“Globals”タブには、“Init Task”および“Exit Task”オプションの欄に `Init` および `Exit` というデフォルトのタスク名が表示されます。サンプルプロジェクトの OS エディタで同じ名前のタスクが定義されているので、ここで他のタスクを選択する必要はありません。“IRQ Handler Task”オプションについてはデフォルトタスクは定義されていないので、ここで定義を行います。

- “IRQ Handler Task”オプションの右側の空欄をダブルクリックします。

  “Task Selection”ダイアログボックスが開きます。



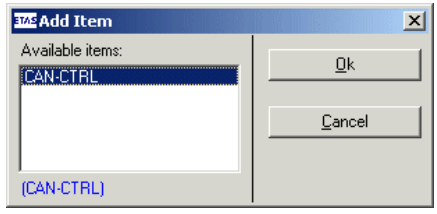
- `Analyze` という名前の Software タスクを選択して **OK** をクリックします。

“ID”オプションの番号は、自動的に設定されます。この `ES1222-CAN` アイテムの他には同じタイプのボードは組み込まれていないため、値は `ID1` となります。

- **Accept** ボタンで設定内容を確定します。

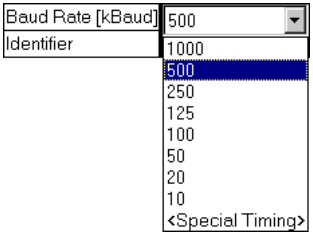
CAN チャンネルを組み込む：

- HWC エディタの“Items”リストから `ES1222-CAN` を選択します。
- **Edit → Add Item...** を選択して、次の階層レベルに組み込むことができるアイテムの一覧を表示します。



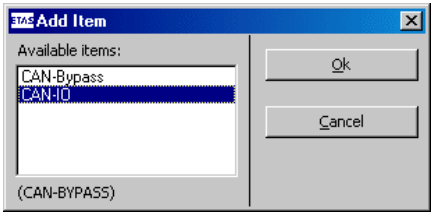
- `CAN-CTRL` というアイテムを選択して **OK** をクリックします。

  このアイテムは、ボード上の各 CAN コントローラに相当します。



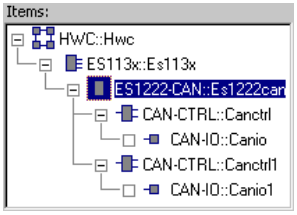
- “Global”タブで“Baud Rate [kBaud]”オプションのデフォルト設定値（1000）をクリックします。

  設定値のフィールドがコンボボックスに変わり、選択できるボーレートの一覧が表示されます。
- **コンボボックスから** 500（kBaud）を選択します。
- **Accept** ボタンで設定内容を確定します。
- “Items”リストから CAN-CTRL を選択します。
- **Edit → Add Item...** を選択して、次の階層レベルで選択可能なアイテムの一覧を開きます。



- CAN-IOというアイテムを選択して**OK**をクリックします。
- さらに、CAN コントローラをもう 1 つ追加し、それに CAN チャンネルを追加します。

これで、サンプルシステムのハードウェア構成に対応するアイテムツリーが完成しました。



11.2.4 ES1222（CAN-IO）の HWC 設定

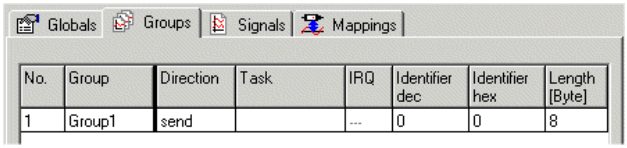
次に、2 つの CAN チャンネルの設定を行います。1 番目のチャンネル（HWC エディタには CANIO::Canio という名前のアイテムとして組み込まれています）がシグナルを送信し、2 番目のチャンネル（CANIO::Canio1）がそれを受信するように、設定します。

サンプルプロジェクトにおいては、各 CANIO アイテムの“Globals" タブには設定する必要のあるオプションはありません。

“Groups”タブ： CAN メッセージを、シグナルグループとして定義します。

シグナルグループ 1 の設定を行う：

- “Items”リストから `CANIO::Canio` を選択します。
- “Groups”タブを選択します。



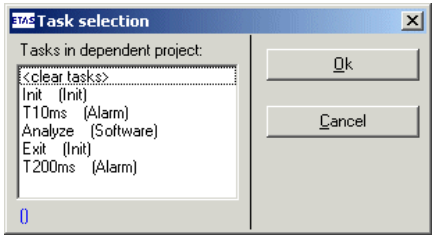
`Group1` という名前のシグナルグループがすでに定義されています。

- `Group1` の行の“Group”カラムをダブルクリックします。

そのフィールドが入力フィールドに変わります。

- `MSG_100` というグループ名を入力します。
- 同じ行の“Task”カラムをクリックします。

“Task selection”ダイアログボックスが開きます。



- `T10ms`というAlarmタスクを選択して**OK**をクリックします。
- 同じ行の“Identifier dec”カラムをクリックします。

- 識別子として、`100` を入力します。

隣の“Identifier hex”カラムに `64` という値がセットされます。

- Accept で設定内容を保存します。

シグナルグループを追加する：

次に、2 番目のシグナルグループを追加します。これも他のタスクに送信されるものです。

- 1 番目のシグナルグループの行の“No.”カラムをクリックします。

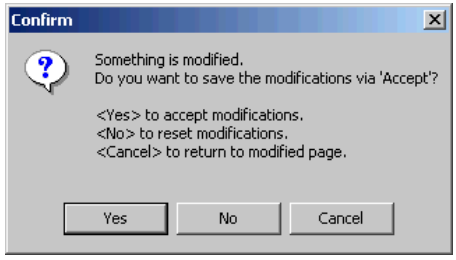
  Edit メニューの、シグナルグループを追加するためのメニューコマンドが有効になります。

- **Edit → Add Row After を選択します。**

または

- “Groups”タブ上でショートカットメニューを開き、**Add Row After** を選択します。

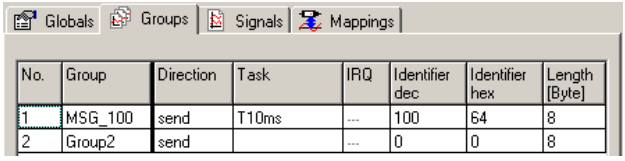
  — 1 番目のグループに対する設定内容がまだ保存されていない場合は、以下のメッセージが表示されます。



  — **OK** でこのメッセージを閉じます。

  — もう一度、**Add Row After** を選択します。

  1 番目のシグナルグループの下に新しい行が追加されます。



No.	Group	Direction	Task	IRQ	Identifier dec	Identifier hex	Length [Byte]
1	MSG_100	send	T10ms	---	100	64	8
2	Group2	send		---	0	0	8

シグナルグループ 2 の設定を行う：

- グループ名を `MSG_101` にします。
- “Task”カラムに `T200ms` という Alarm タスクを選択します。

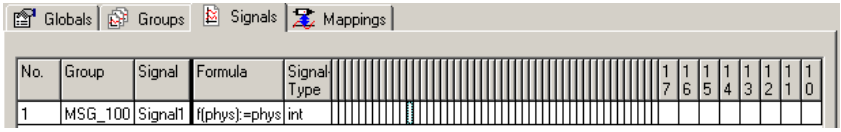
- “Identifier dec”カラムに 101 という値を入力します。

  隣の“Identifier hex”カラムに 65 という値がセットされます。
- Accept で設定内容を保存します。

“Signals”タブ： このタブで各 CAN シグナルを定義します。初期状態においてすでに 1 つのシグナルが存在していますが、サンプルプロジェクトでは 2 つの数値シグナルと 1 つの論理シグナルが必要なので、まず最初にシグナルを 2 つ追加します。

シグナルを追加する：

- “Signals”タブを選択します。



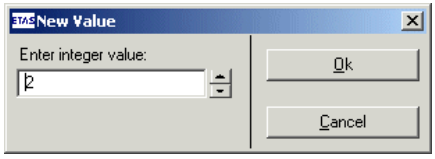
- 1 行目のシグナルの“No.”カラムをクリックして、選択します。

  Edit メニューの、シグナルを追加するためのメニューコマンドが有効になります。
- **Edit → Add Multiple Rows After を選択します。**

または

- “Groups”タブ上でショートカットメニューを開き、**Add Row After** を選択します。

  “New Value”ダイアログボックスが開きます。

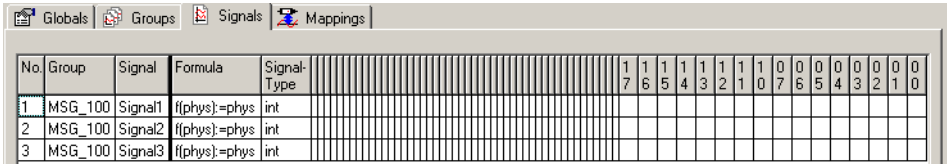


- 入力フィールドに 2 を入力します。

  上下の矢印ボタンを使用して入力フィールドの数字を変更することもできます。
- **OK** をクリックします。

  選択されていた 1 行目のシグナルの下に新しい行が 2 行追加されます。新しいシグナルには 1 行目のシグナルと同じグループに割り当てられ、シグ

ナル名は、Signal<n> というように最後の数字がカウントアップされた名前が割り当てられます。



（上の図は、設定項目全体が見えるように、ビットマトリックス（178 ページの「7654321.. (Bit matrix)」を参照してください）のうちでこのサンプルプロジェクトに必要ないカラムが省略されています。）

論理シグナル、および 2 つの数値シグナルのうちの 1 つは 1 番目のシグナルグループで転送し、残りの 1 つの数値シグナルは 2 番目のグループで転送します。

論理シグナルを設定する：

- 1 行目のシグナルの“Signal”カラムをダブルクリックします。
- シグナル名を vcansend1log に変更します。

- 同じ行の“Signal-Type”カラムをダブルクリックします。

  そのフィールドがコンボボックスに変わり、選択できるシグナルタイプの一覧が表示されます。
- コンボボックスで bool を選択します。

- 同じ行の“00”カラム（ビットマトリックスの最後のカラム）をダブルクリックします。

  そのフィールドに 1 がセットされます。これで、このシグナルが、1 番目の CAN メッセージのバイト 0 のビット 0 を使用して転送されるように設定されました。

1 番目の数値シグナルを設定する：

- 2 行目のシグナルの名前を vcansend2cont に変更します。
- 同じ行の“Signal-Type”カラムで uint 型を選択します。
- 同じ行の“1 0”カラムをダブルクリックします。

  そのフィールドに 1 がセットされます。

- 同じ行の“17”カラムをダブルクリックします。

  そのフィールドに 1 がセットされます。
- **<Shift>** キーを押しながら、もう一度“17”カラムをクリックします。

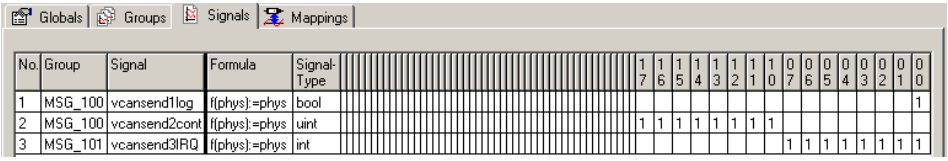
  “10”から“17”までのすべてのカラムに 1 がセットされます。これで、このシグナルが、1 番目の CAN メッセージのバイト 1 のビット 0~7 を使用して転送されるように設定されました。

2 番目の数値シグナルを設定する：

- 3 行目のシグナルの名前を `vcansend3IRQ` に変更します。
- 同じ行の“Group”カラムで `MSG_101` グループを選択します。
- 同じ行の“Signal-Type”カラムで `uint` 型を選択します。
- 同じ行の“00”～“07”カラムに 1 をセットします。

  これで、このシグナルが、2 番目の CAN メッセージの バイト 0 のビット 0~7 を使用して転送されるように設定されました。
- **Accept** をクリックして設定内容を保存します。

3 つのシグナルをすべて設定すると、“Signals”タブは以下のようになります。

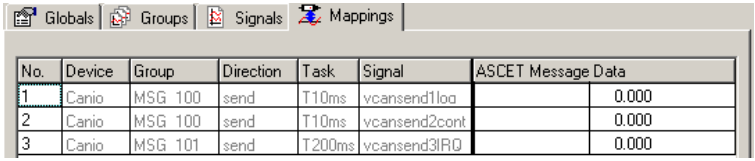


“Mappings”タブ： このタブで、各 CAN シグナルを、プロジェクト内に定義されている ASCET メッセージに割り当てます。各シグナルの“ASCET Message”カラムをクリックするとメッセージを選択するダイアログボックスが開き、このダイアログボックスには、*Exported* 属性を持つメッセージのうち、以下のようにシグナルグループの転送方向（*send* または *receive*）に対応するメッセージのみが表示されます。

- 転送方向 = receive → 受信メッセージ
- 転送方向 = send → 送信メッセージ

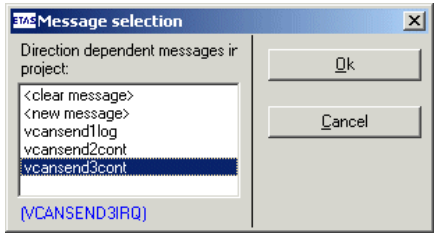
ASCET メッセージをマニュアル操作で割り当てる：

- “Mappings”タブを選択します。



- 3 行目のシグナルの“ASCET Message”カラムをクリックします。

  “Message selection”ダイアログボックスが開きます。



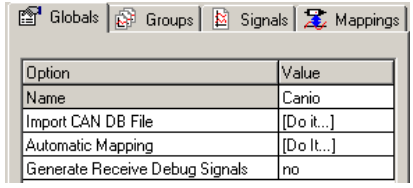
  ASCET プロジェクトに定義されているすべての送信メッセージが表示されます。

- vcansend3contメッセージを選択して**OK**をダブルクリックします。
- **Accept** をクリックして設定内容を保存します。

1 行目と 2 行目のシグナルについては、同名の ASCET メッセージがプロジェクト内に定義されています。上記の方法でそれらを 1 つずつ割り当てることもできますが、以下のように自動的に割り当てることも可能です。

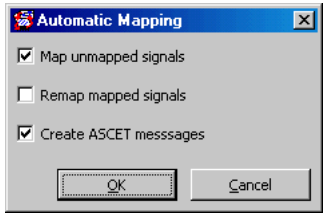
ASCET メッセージを自動的に割り当てる：

- “Globals”タブを選択します。



- “Automatic Mapping”オプションの“Value“カラムをダブルクリックします。

  “Automatic Mapping”ダイアログボックス が開きます。

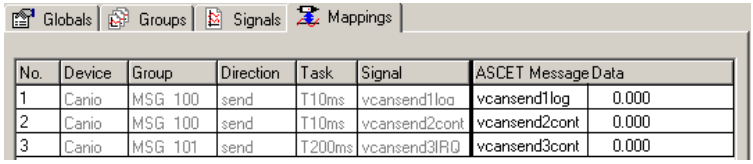


  詳しい説明は、173 ページの「Automatic Mapping」を参照してください。

- **Map unmapped signals** および **Create ASCET messages** オプションをオンにします。

  `vcansend1log` シグナルと `vcansend2cont` シグナルが、それぞれ同名の ASCET メッセージに自動的に割り当てられます。
  すでにマニュアル操作でメッセージに割り当てられている `vcansend3IRQ` シグナルは、影響を受けません。

“Mappings”タブ上のすべてのシグナルが、以下のように ASCET メッセージに割り当てられました。



No.	Device	Group	Direction	Task	Signal	ASCET Message	Data
1	Canio	MSG_100	send	T10ms	vcansend1log	vcansend1log	0.000
2	Canio	MSG_100	send	T10ms	vcansend2cont	vcansend2cont	0.000
3	Canio	MSG_101	send	T200ms	vcansend3IRQ	vcansend3cont	0.000

これで、1 番目の CAN チャンネルの設定がすべて終了しました。

チャンネル 2

“Groups” タブ： CAN メッセージを、シグナルグループとして定義します。

シグナルグループ 1 の設定を行う：

- “Items”リストから `CANIO::Canio1` を選択します。

- “Groups”タブを選択します。

  タブ上の各設定オプションには、チャンネル 1 と同様の初期設定が行われています。
- 1番目のシグナルグループの名前をMSG_100にします。

チャンネル 2 はシグナルの受信を行うので、シグナルグループの転送方向を変更する必要があります。

- シグナルグループの“Direction”カラムをダブルクリックします。

  そのフィールドがコンボボックスに変わり、選択できる方向の一覧が表示されます。
- receive を選択します。
- 同じ行の“Task”カラムで T10ms という Alarm タスクを選択します（283 ページを参照してください）。
- 同じ行の“Identifier dec”カラムに 100 という値を入力します。

  隣の“Identifier hex”カラムには 64 という値がセットされます。
- Accept で設定内容を保存します。

次に、284 ページの方法で、割り込みモードで転送される 2 番目のシグナルグループを追加します。

シグナルグループ 2 の設定を行う：

- 2番目のシグナルグループ名をMSG_101にします。
- “Direction”カラムで、転送方向として receive を選択します。

- 同じ行の“IRQ”カラムをダブルクリックします。

  そのフィールドがコンボボックスに変わり、選択できるオプションの一覧が表示されます。
- yes を選択します。

  自動的に“Task”カラムの内容がリセットされ、入力がロックされます。“Prescaler”カラムには、自動的に 1 がセットされます。

No.	Device	Group	Direction	Task	IRQ	Identifier dec	Identifier hex	Length [Byte]	Activated Task	Pre-scaler
1	Canio1	MSG_100	receive	T10ms	no	100	64	8		---
2	Canio1	MSG_101	receive	---	yes	0	0	8		1

- “Identifier dec”カラムに 101 という値を入力します。

  隣の“Identifier hex”カラムに 65 という値がセットされます。
- Accept で設定内容を保存します。

“Signals”タブ： このタブで各 CAN シグナルを定義します。初期状態においてすでに 1 つのシグナルが存在しているので、285 ページの方法で、2 つのシグナルを追加します。

論理シグナル、および 2 つの数値シグナルのうちの 1 つは 1 番目のシグナルグループで転送し、残りの 1 つの数値シグナルは 2 番目のグループで転送します。

論理シグナルを設定する：

- 1 行目のシグナルの名前を vcanrec1log に変更します。
- 同じ行の“Signal-Type”カラムで bool 型を選択します。
- 同じ行の“0 0”カラムをクリックして 1 をセットします（286 ページ参照参照）。

  これで、このシグナルが、1 番目の CAN メッセージのバイト 0 のビット 0 を使用して転送されるように設定されました。

1 番目の数値シグナルを設定する：

- 2行目のシグナルの名前をvcanrec2contに変更します。
- 同じ行の“Signal-Type”カラムで uint 型を選択します。
- 同じ行の“1 0”～“1 7”カラムに 1 をセットします（286 ページ参照）。

  これで、このシグナルが、1 番目の CAN メッセージのバイト 1 のビット 0~7 を使用して転送されるように設定されました。

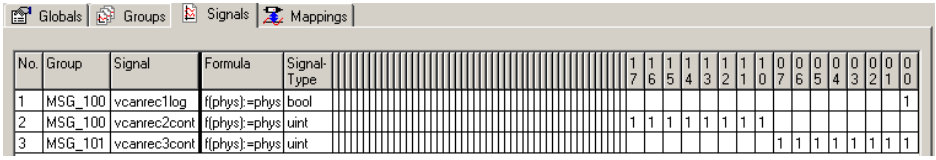
2 番目の数値シグナルを設定する：

- 3行目のシグナルの名前をvcanrec3contに変更します。
- 同じ行の“Group”カラムで MSG_101 グループを選択します。
- 同じ行の“Signal-Type”カラムで uint 型を選択します。

- 同じ行の“0 0”～“0 7”カラムに 1 をセットします。

  これで、このシグナルが、2 番目の CAN メッセージのバイト 0 のビット 0~7 を使用して転送されるように設定されました。
- **Accept** をクリックして設定内容を保存します。

3 つのシグナルをすべて設定すると、“Signals”タブは以下のようになります。



3 つのシグナルと同名のメッセージがプロジェクト内に定義されているため、メッセージの割り当てはすべて自動的に行えます。

ASCET メッセージを自動的に割り当てる：

- “Globals”タブを選択します。
- “Automatic Mapping”オプションの“Value“カラムをダブルクリックします。

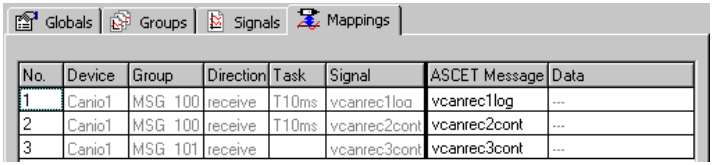
  “Automatic Mapping”ダイアログボックス が開きます。

  詳しい説明は、173 ページの「Automatic Mapping」を参照してください。
- **All** をクリックします。

  3 つのシグナルが、それぞれ同名の ASCET メッセージに自動的に割り当てられます。

また、288 ページのように、“Mappings”タブで個別に割り当てを行うことも可能です。

“Mappings”タブ上のすべてのシグナルが、以下のように ASCET メッセージに割り当てられました。



これで、2 番目の CAN チャンネルの設定がすべて終了しました。

11.2.5 ハードウェアコンフィギュレーションの保存

作成したハードウェアコンフィギュレーションは、プロジェクトのファイルコンテナ内に保存したり、または DOS ファイル（*.HWC）として保存することができます。

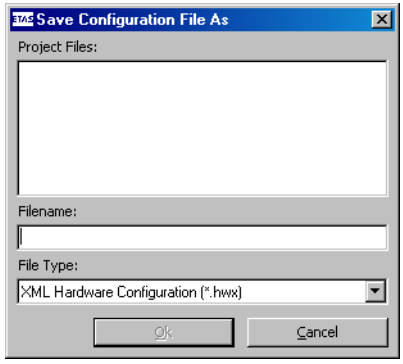
ハードウェアコンフィギュレーションをファイルコンテナに保存する：

- **Save** ボタンをクリックします。

または

- HWC エディタメニューから **File → Save** を選択して、ハードウェアコンフィギュレーションを保存します。

  現在のコンフィギュレーションを初めて保存する際には、“Save Configuration File” が開きます。



- “File Type” コンボボックスから、`Hardware Configuration (*.hwx)` というファイルタイプを選択します。
- ファイル名を入力します。

注記

サンプルプロジェクトの参考用コンフィギュレーションは、`ES1222.hwx` という名前で保存されています。このファイルを上書きしないように注意してください。

- **OK** をクリックします。

  ハードウェアコンフィギュレーションが、ファイルコンテナに保存されます。この内容は、“Files” タブに表示されます。

ハードウェアコンフィギュレーションを DOS ファイルに保存する：

- HWC エディタで、**File → Export** メニューを選択します。
- “Export Hardware Configuration” ダイアログボックスで、ハードウェアコンフィギュレーションを保存するファイルタイプ、およびファイルのパスと名前を指定します。

  ハードウェアコンフィギュレーションが、指定されたファイルに保存されます。このファイルは後に、HWC エディタから **File → Import** でインポートすることができます。

11.2.6 HWC モジュールのコード生成

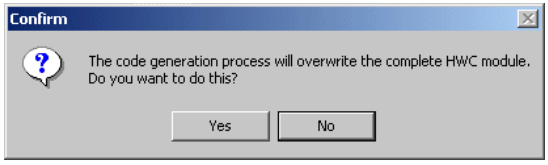
次に、HWC モジュールのコード生成処理を行います。

HWC モジュールのコードを生成する：

- HWC エディタの **Extras → Generate Code → For Current Experiment** メニューコマンドを選択します。

または

- **Generate Code for Current Experiment** ボタンをクリックします。
- HWC モジュールの上書きを確定します。



- プロジェクトのコード生成オプションで選択されている実験タイプに合わせて、HWC モジュールの C コードが生成されます。

また、以下のようにして他のタイプの実験用のコードを生成することもできます。

- HWCエディタで**Extras → Generate Code → For Phys. And Quant. Experiment** を選択して、物理実験および量子化実験用のコードを生成します。
- HWCエディタで**Extras → Generate Code → For Phys., Quant. And Impl. Experiment** を選択して、物理実験、量子化実験、および実装実験用のコードを生成します。

11.2.7 サンプルプロジェクトの実験

前項までで RTIO に関する設定がすべて完了したので、HWC エディタを閉じ、次にプロジェクトエディタで実験ターゲット用の標準的なコードを生成します。

注記

HWC モジュールをプロジェクトエディタのグラフィック表示に含めることはお勧めできません。HWC モジュールは RTIO のコード生成が行われるたびに変わるため、予期しない形で表示されてしまうためです。

実験ターゲット用の実行コードを生成する：

- プロジェクトエディタで **Component → Build** を選択して、プロジェクト全体のコード生成を行います。
- **Component → View Generated Code** を選択して生成されたコードを表示し、内容を確認します。

- 次の図のように、ES1222 の CAN ポート、CAN1 および CAN2 を、適切なケーブルとターミネータで接続します。

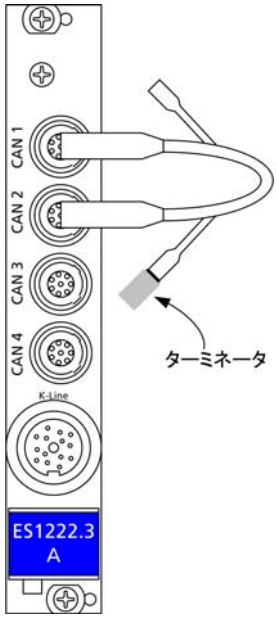


図 11-3 ES1222 にターミネータ付きケーブルを接続する

注記

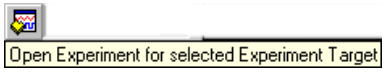
ターミネータを使用しないと、通信は行なわれません。

- “Experiment Target”コンボボックスから `Online (RP)` を選択します。

  `Offline (RP)` は、ターゲット上でオフライン実験を行うための設定です。

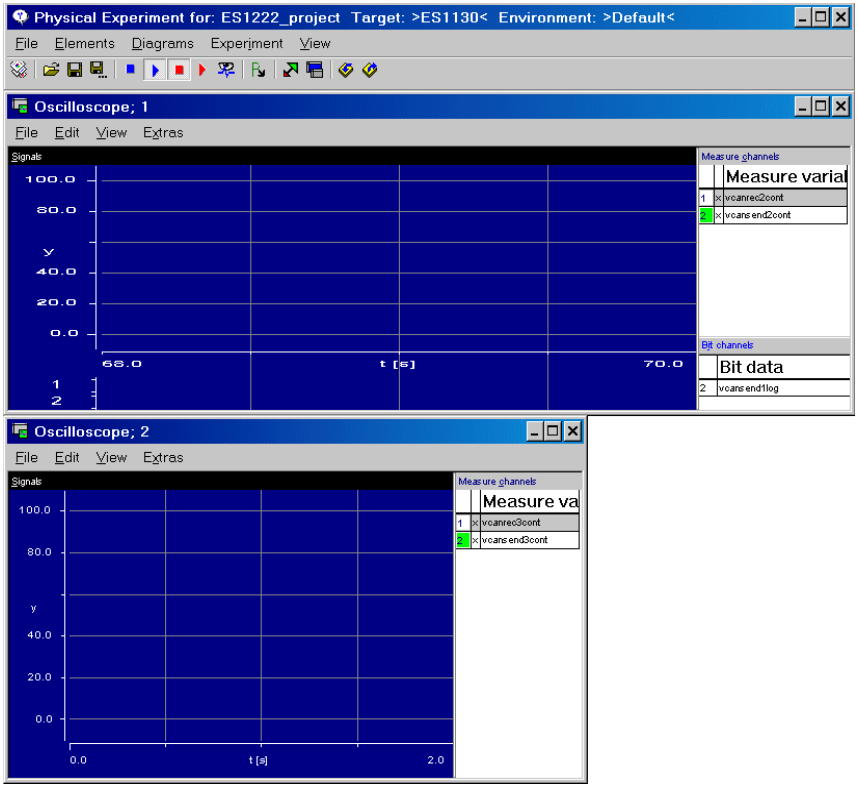
- **Component → Open Experiment** を選択します。

または



- **Open Experiment for selected Experiment target** ボタンをクリックします。

  “Physical Experiment”ウィンドウに、2 つのオシロスコープを含む実験環境が開きます。



- “Physical Experiment”ウィンドウで、**Experiment → Start ERCOS** を選択します。

または

- **Start ERCOS** ボタンをクリックします。

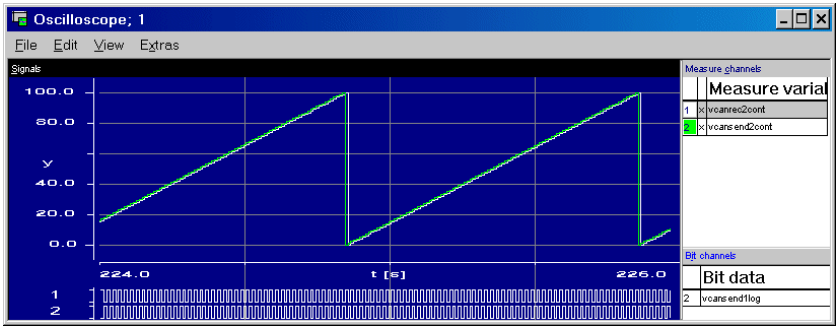
  オペレーティングシステムが起動し、モデルが実行されます。

- “Physical Experiment”ウィンドウで、**Experiment → Start Measurement** を選択します。

または

- **Start Measurement** ボタンをクリックします。
- ASCET メッセージの値が 2 つのオシロスコープ上に表示されます。

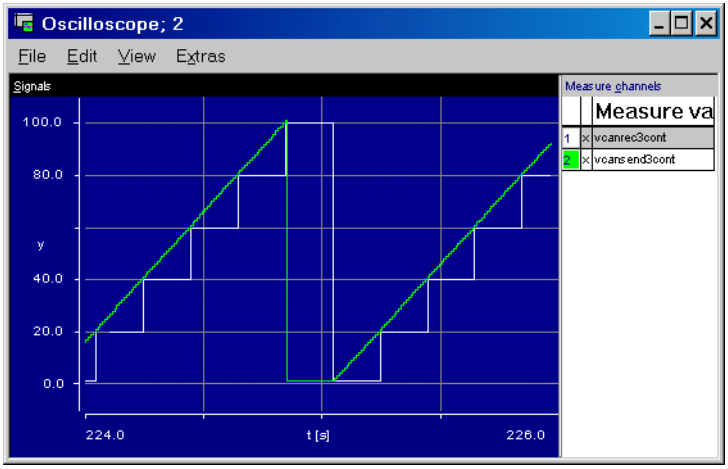
上側のオシロスコープの“Signals”エリアに、送信メッセージ vcansend2cont（左側の緑色のライン）と受信メッセージ vcanrec2cont（右側の白いライン）が表示され、ビット表示部には受信メッセージ vcanrec1log（上側のライン）と送信メッセージ vcansend1log（下側のライン）が表示されます。



表示される信号の値は、以下のようにして処理されています。

- 送信メッセージ vcansend2cont の値は、10ms 周期で T10ms タスクの calcCANLoop プロセスにおいてセットされ、同タスクの CanioMSG100_T10ms_HWCL プロセスによって送信されます。
- 送信された値は、受信メッセージ vcanrec2cont 経由で T10ms タスクの CanioMSG100_T10ms_HWCF プロセスによって読み込まれ、ここで 1 周期分の遅延が発生します。

下側のオシロスコープには、送信メッセージ vcansend3cont（細かいステップの緑色のライン）と受信メッセージ vcanrec3cont（大きなステップの白いライン）が表示されます。



表示される信号の値は、以下のようにして処理されています。

- 送信メッセージ `vcansend3cont` の値は、`10ms` 周期で `T10ms` タスクの `calcCANLoop` プロセスにおいてセットされるので、上のオシロスコープの `vcansend2cont` と同様に連続的なラインとなります。この値は、`200ms` 周期で `T2000ms` タスクの `CanioMSG101_T200ms_HWCL` プロセスによって送信されます。
- 送信された値は割り込みモードで受信されます。つまり、`Analyze` タスクからトリガが発行された時にメッセージ `vcanrec3cont` 経由で受信されます。そのため、表示されるラインは段階的に変化します。

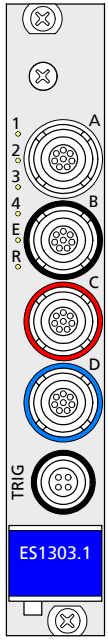
11.3 チュートリアル - ES1303

この章では、RTIO パッケージを使用して、ES1303 AD ボードを ASCET プロジェクトに組み込みます。組み込みを行う前に、6「準備」の章に書かれている準備作業を行う必要があります。また RTIO チャンネルとモデル間のデータ交換を行うには、モデル内において *Exported* 属性で定義されているメッセージが必要です。

ES1303 ボードの場合、メッセージの受信のみを行い、送信は行いません。そのためここでは、実際の実験での操作例は示されていません。

11.3.1 ES1303 ハードウェア

以下の図は、ES1303 A/D コンバータボードのフロントパネルの概観です。



A ～ D の入力ポートにそれぞれ 4 チャンネルのアナログ入力を接続できます。
TRIG 入力はトリガ信号用です。

ES1303 の入力電圧範囲

ES1303 ボードの入力電圧範囲は、HWC エディタで ±10V または ±60V に設定できます。

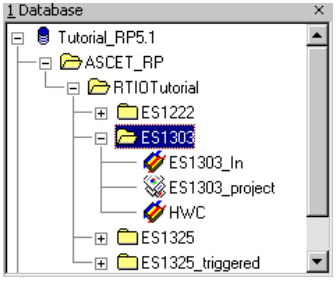
複数の ES1303 ボードの使用

1 台の ES1000 システム内で、最大 4 枚の ES1303 ボードを使用できます。

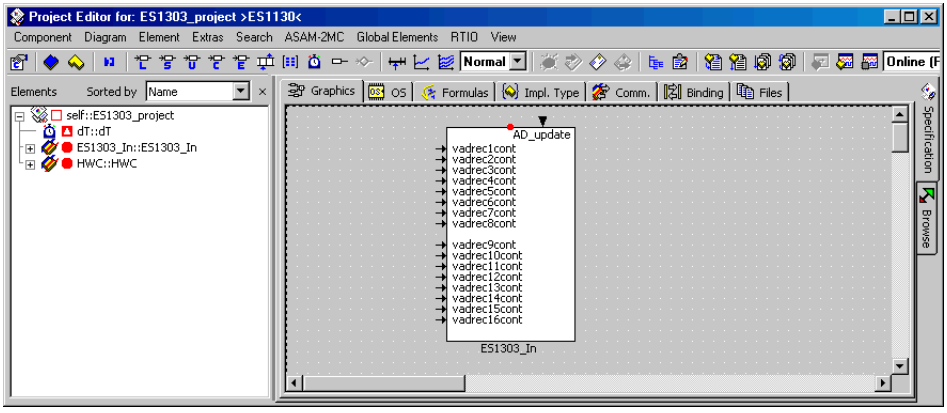
11.3.2 サンプルプロジェクト

例題を開く：

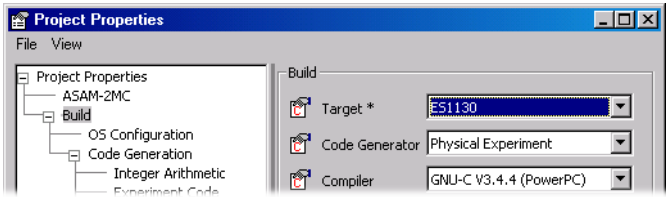
- コンポーネントマネージャで、ASCET_RP/RTIOTutorial/ES1303 フォルダを選択します。



- ES1303_project というプロジェクトを開きます。



- **Project Properties** ボタンをクリックします。
  “Project Properties”ウィンドウが開きます。



- “Build”ノードで以下のオプションを設定します。
  `Target: >ES1130< または >ES1135<`
  `Compiler: GNU-C * (PowerPC)`

注記

RTIO との通信に使用できるのは、“Exported”として宣言されたメッセージのみです。

サンプルプロジェクトには、ES1303 の RTIO 統合に必要な要素（タスク、メッセージ、HWC モジュール）がすでに含まれています。

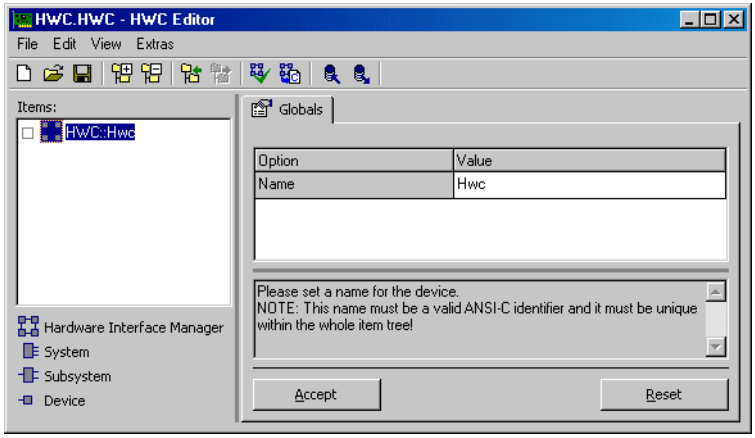
注記

通常、ハードウェアコンフィギュレーションを作成するには、まず HWC という C コードモジュールを作成し、それをプロジェクトに組み込んでからその内容を編集しますが、このチュートリアルでは、すでにこの HWC モジュールがプロジェクトに組み込まれています。

HWC エディタを開く：

- プロジェクトエディタで **RTIO → Open Editor** を選択します。

  HWC エディタが開きます。



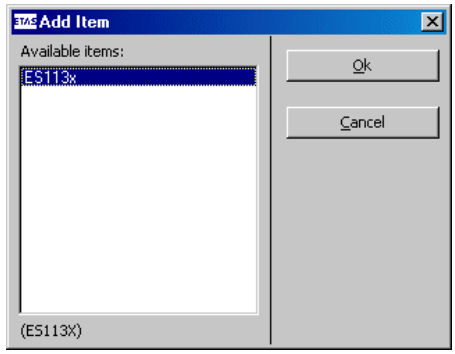
ハードウェアコンフィギュレーション（HWC）を作成する：

各ハードウェアは、アイテムリスト内にツリー構造の形で定義します。このツリーのルートには、必ず HWC というアイテムが存在します。

- HWC エディタで、**Edit → Add Item...** を選択します。

または

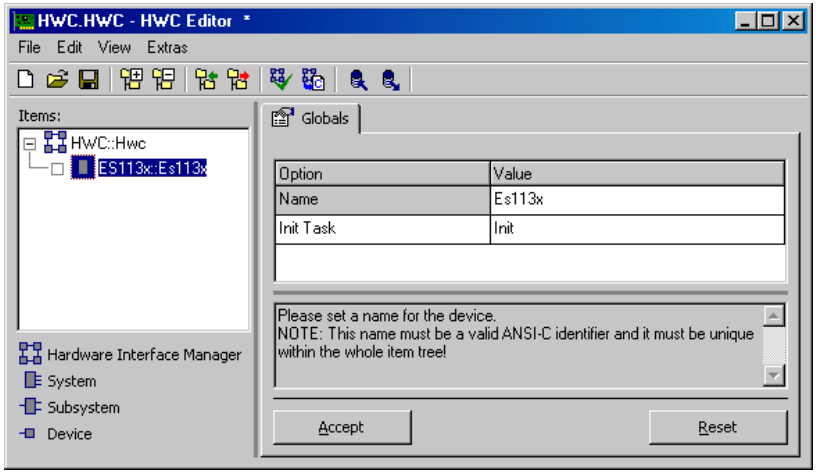
- **Add Item...** ボタンをクリックします。
  “Add Item”ダイアログボックスが開きます。



  現在選択されているアイテムの次の階層レベルに追加できるアイテムの一覧が表示されます。

- ES113x を選択します。
  このアイテムは、ES1000.x システムに組み込まれる PowerPC プロセッサボード ES1130 または ES1135 に相当します。

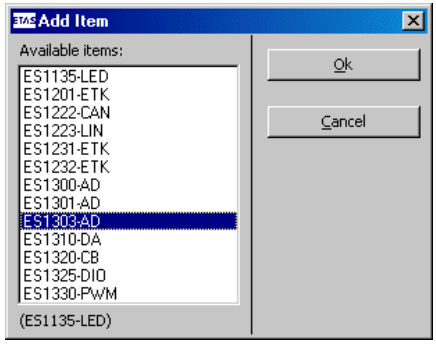
- **OK** をクリックします。
  “Items”リストに ES113x が追加されます。



- “Items”リストから ES113x を選択します。

“Globals”タブには、“Init Task”オプションの欄に `Init` というデフォルトのタスク名が表示されます。サンプルプロジェクトの OS エディタで同じ名前の Init タスクが定義されているので、ここで他のタスクを選択する必要はありません。

- 再度 **Edit → Add Item...** を選択して、次の階層レベルで選択可能なアイテムの一覧を開きます。



- `ES1303-AD` というアイテムを選択します。

  このアイテムは、ES1303 A/D インターフェースに相当します。
- **OK** をクリックします。

これで、サンプルシステムのハードウェア構成に対応するアイテムツリーが完成しました。

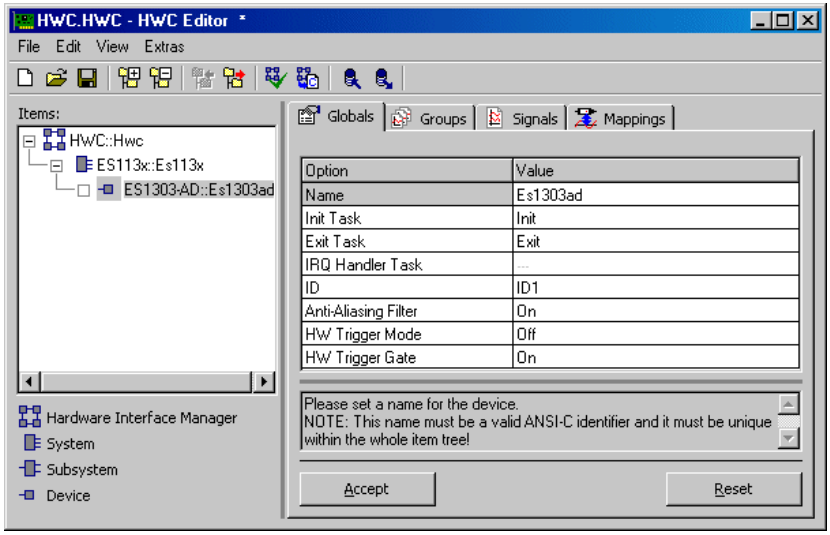
11.3.4 ES1303 の HWC 設定

次に、“Globals”タブでタスクの設定を行います。

“Globals”タブの設定を行う：

- アイテムリストから、ES1303-AD というアイテムを選択します。

- “Globals”タブを選択します。



“Globals”タブには、“Init Task”および“Exit Task”オプションの欄に `Init` および `Exit` というデフォルトのタスク名が表示されます。サンプルプロジェクトのOS エディタで同じ名前のタスクが定義されているので、ここで他のタスクを選択する必要はありません。

“IRQ Handler Task”オプションについてはデフォルトタスクは定義されていません。ハードウェアトリガを使用する（“HW Trigger Mode”オプション）場合は、トリガを受け取る割込み処理タスクをここに定義します。

“ID”オプションの番号は、自動的に設定されます。この `ES1303-AD` アイテムの他には同じタイプのボードは組み込まれていないため、値は `ID1` となります。

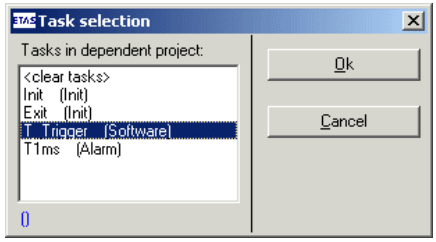
- 入力シグナルの帯域幅に制限を与えるには、“Anti-Aliasing Filer”オプションを `On` に設定します。
- “HW Trigger Mode”オプションで、トリガ信号のどのエッジで測定を開始するかを指定します。（例： `Falling Edge`）
- “HW Trigger Gate”オプションで、第 2 のトリガ信号を使用するかどうかを指定します。

注記

ポーリングモード（“Groups”タブの“IRQ”オプションが `No` の場合）においては、“HW Trigger Mode”および“HW Trigger Gate”オプションは無効です。

- “IRQ Handler Task”オプションの右側の空欄をクリックします。

  “Task Selection”ダイアログボックスが開きます。



- `T_Trigger`というSoftwareタスクを選択してOKをクリックします。

  各オプションの設定内容は、以下の例のようになります。

オプション	値
Name	`ES1303ad`
Init Task	`Init`
Exit Task	`Exit`
IRQ Handler Task	`T_Trigger`
ID	`ID1`
Anti-Aliasing Filter	`On`
Trigger Mode	`Falling Edge`
Trigger Gate	`On`

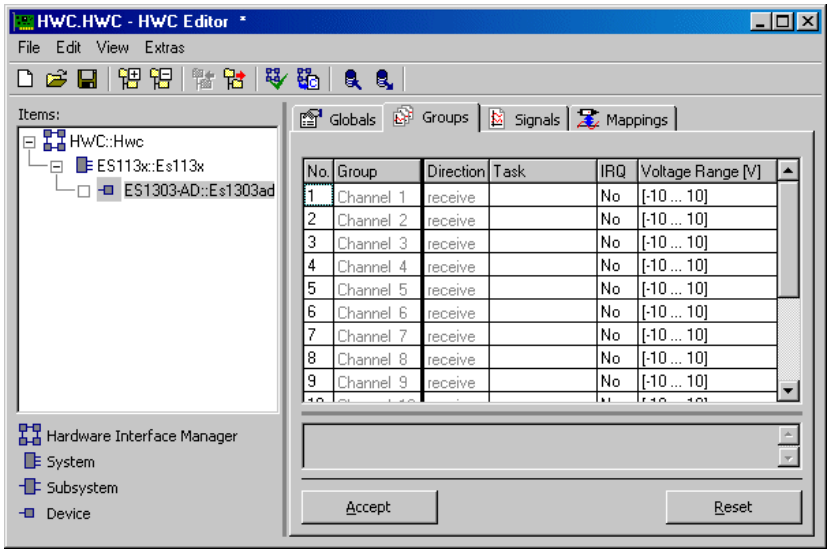
- すべての設定が終了したら、**Accept** ボタンで設定内容を保存します。

次の“Groups”タブでは、シグナルグループに関する設定を行います。

ES1303 ボードでは、各チャンネルごとに 1 つのシグナルグループが固定的に使用されます。ここでは、タスク割り当て、入力電圧範囲、および受信モード（割込みまたはポーリング）に関する設定を行います。

“Groups”タブの設定を行う：

- “Groups”タブを選択します。



使用するすべてのシグナルについて、以下の設定を行います。

1. 受信タイプの選択：

- 割り込みモードで読み込みを行うシグナルの“IRQ”カラムをダブルクリックします。

  コンボボックスが開きます。
- コンボボックスから Yes を選択します。

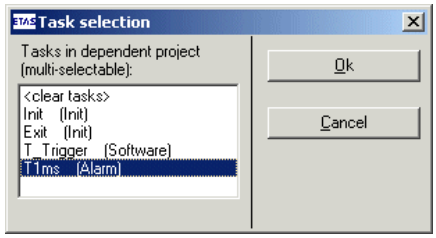
  選択された設定が、“IRQ”カラムに表示されます。同時に、“Task”カラムがリセットされ、入力がブロックされます。

No.	Group	Direction	Task	IRQ	Voltage Range [V]
1	Channel 1	receive	---	Yes	[-10 ... 10]
2	Channel 2	receive		No	[-10 ... 10]

- またポーリングモードで受信を行うには、“IRQ”カラムを No（デフォルト設定）のままにしておきます。

2. タスクの割り当て（ポーリングモードのシグナルのみ）：
   - タスクを割り当てるシグナルの“Task”カラムをクリックして、“Task Selection”ダイアログボックスを開きます。

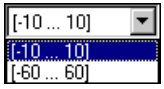
- タスク選択リストから、T1ms というタスクを選択します。このタスクがデータ転送に使用されます。



タスク選択リストから、複数のタスクを選択することもできます。ES1303 の場合、通常は Alarm タスクを選択します。

- **OK** をクリックします。
- 選択されたタスクが“Task”カラムに表示されます。

3. 電圧範囲の選択：



- 入力信号の電圧範囲を選択するシグナルの“Voltage Range”カラムをクリックします。

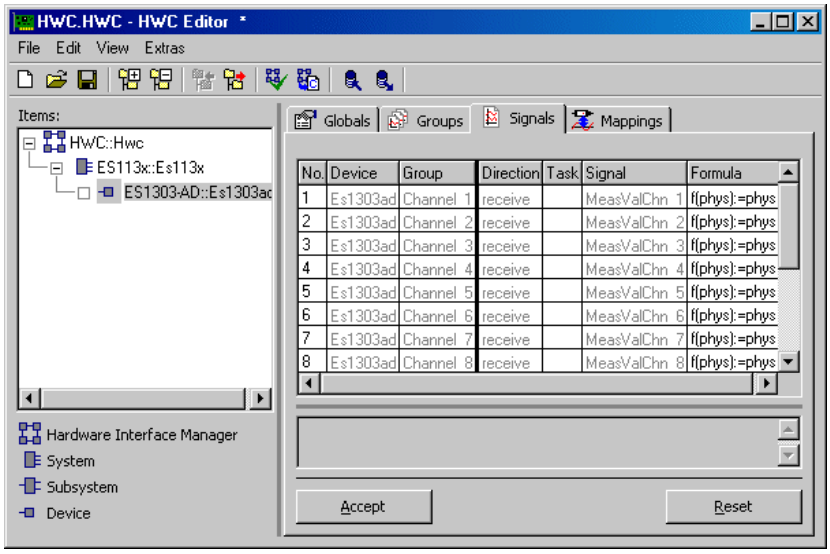
  コンボボックスが開きます。
- コンボボックスから電圧範囲を選択します。

  選択された設定が、“Voltage Range”カラムに表示されます。

No.	Group	Direction	Task	IRQ	Voltage Range [V]
1	Channel 1	receive	---	Yes	[-10 ... 10]
2	Channel 2	receive		No	[-60 ... 60]

- すべての設定が終了したら、**Accept** ボタンで設定内容を保存します。

“Signals”タブには各シグナルの設定オプションが含まれます。ES1303 ボードの場合、特殊なオプションはないので、ここで行う設定作業はありません。タブを開いて、現在の設定内容の確認だけを行ってください。

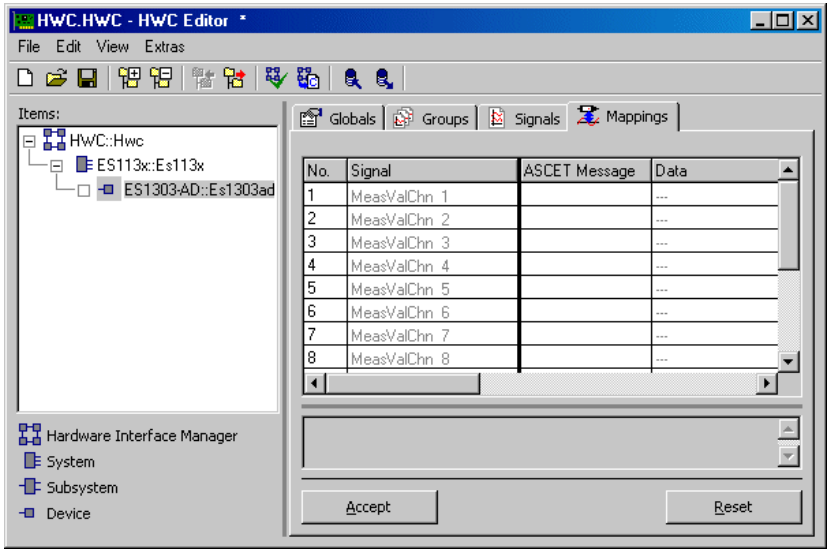


“Mappings”タブでは、プロジェクト内に定義されている ASCET メッセージに各シグナルを割り当てます。各シグナルの“ASCET Message”カラムをクリックするとメッセージを選択するダイアログボックスが開き、このダイアログボックスには、*Exported* 属性を持つメッセージのうち、そのシグナルグループの転送方向（*send* または *receive*）に対応するメッセージのみが表示されます。

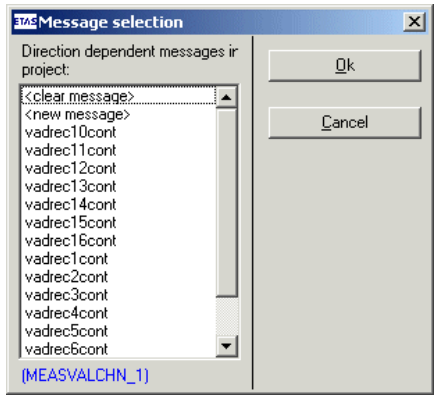
- 転送方向 = receive → 受信メッセージ
- 転送方向 = send → 送信メッセージ（ES1303 では使用されません）

“Mappings”タブの設定を行う：

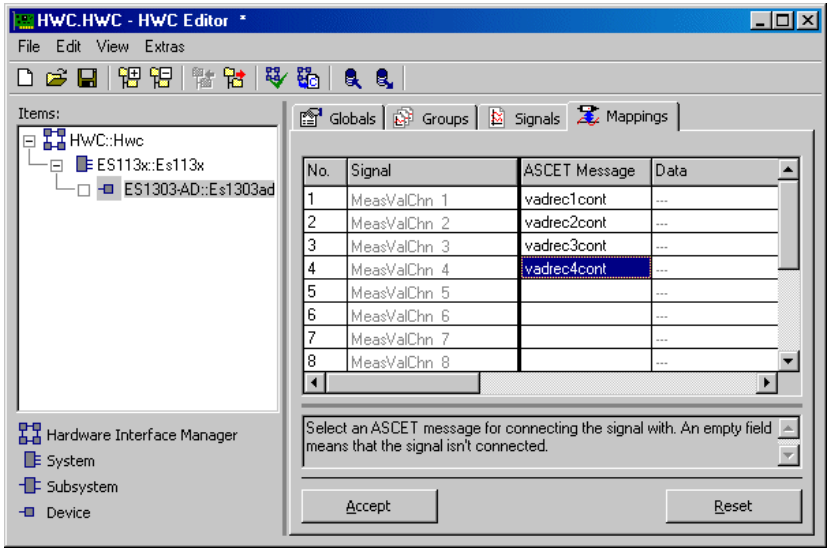
- “Mappings”タブを選択します。



- いずれかの行の“ASCET Message”カラムをダブルクリックして“Message selection”ダイアログボックスを開きます。



このダイアログボックスで、16 個の受信メッセージを、対応するシグナルに 1 つずつ割り当てます。



- **Accept** ボタンをクリックして、設定内容を保存します。

11.3.5 ハードウェアコンフィギュレーションの保存

作成したハードウェアコンフィギュレーションは、プロジェクトのファイルコンテナ内に保存したり、または DOS ファイル（`*.HWX`）として保存することができます。293 ページの 11.2.5 項を参照してください。

注記

サンプルプロジェクトの参考用コンフィギュレーションは、`ES1303.hwx` という名前で保存されています。このファイルを上書きしないように注意してください。

11.3.6 HWC モジュールのコード生成

続いて、HWC モジュールのコード生成を行います。コード生成の方法は、294 ページの 11.2.6 項を参照してください。

11.3.7 サンプルプロジェクトの実験

前項までで RTIO に関する設定がすべて完了したので、HWC エディタを閉じ、次にプロジェクトエディタで実験ターゲット用の標準的なコードを生成します。

注記

HWC モジュールをプロジェクトエディタのグラフィック表示に含めることはお勧めできません。HWC モジュールは RTIO のコード生成が行われるたびに変わるため、予期しない形で表示されてしまうためです。

実験ターゲット用の実行コードを生成する：

- プロジェクトエディタで **Component → Build** を選択して、プロジェクト全体のコード生成を行います。
- **Component → View Generated Code** を選択して生成されたコードを表示し、内容を確認します。

オンライン実験を行う：

- プロジェクトエディタの“Experiment Target”コンボボックスから `Online (RP)` を選択します。

  `Offline (RP)` は、ターゲット上でオフライン実験を行うための設定です。
- **Component → Open Experiment** を選択します。
- 実験環境ブラウザで、`Tutorial` という環境を選択します。

- “Physical Experiment”ウィンドウ で、**Experiment → Start ERCOS** を選択します。
- “Physical Experiment”ウィンドウで、**Experiment → Start Measurement** を選択します。

  ES1303 のアナログ測定値が、ASCET のオシロスコープに表示されます。

11.4 チュートリアル - ES1325（トリガを使用しない）

トリガを使用しない ES1325 サンプルプロジェクトには、ボードの主要な機能がすべて含まれています。モデルのブロックダイアグラムは、以下のように、各ファンクションがグラフィック階層で記述されています。

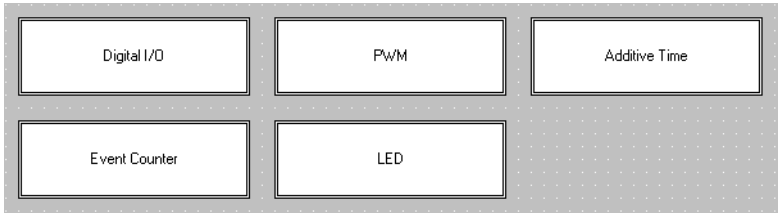


図 11-4 ES1325 ― モデル概観

以下に、各ブロックの機能をまとめます。

デジタル I/O: 入力チャンネル 3 からシグナルが読み取られ、出力チャンネル 3 に送られます。このシグナルは、読み取り時に反転され、Low レベルが“active”ステートとなりますが、送信時には High レベルが“active”ステートになります（243 ページの「Active State」を参照してください）。

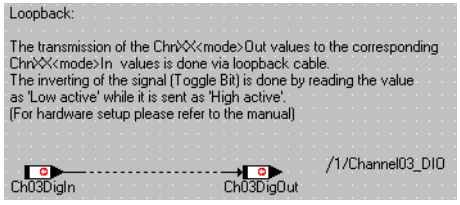


図 11-5 ES1325 ― デジタル I ／ O

PWM: チャンネル 4 は、アクティブ時間と非アクティブ時間が一定である静的な PWM シグナルを出力し、チャンネル 5 のシグナルは、デューティーサイクルが鋸歯シグナルにより決まる動的な PWM シグナルを出力します。

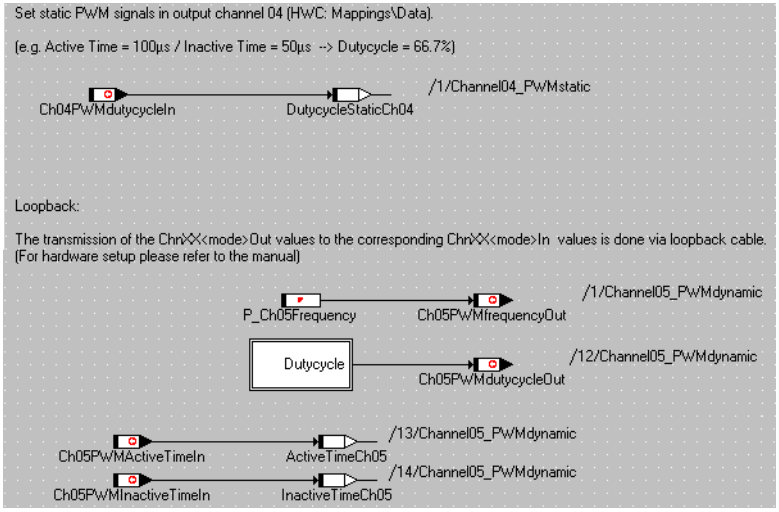


図 11-6 ES1325 — PWM

アクティブタイムの合計： チャンネル 5 の PWM 出力シグナルは、出力チャンネル 6 からも送信されます。このシグナルを入力チャンネル 6 から読み込み、タスク実行時におけるシグナルのアクティブフェーズの合計時間を算出します。

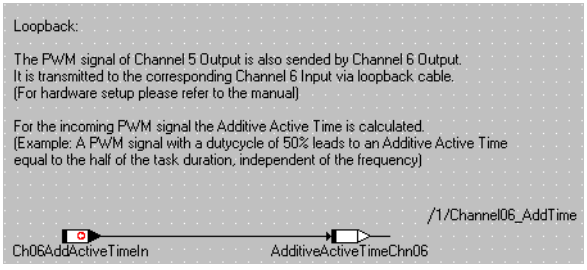


図 11-7 ES1325 — アクティブタイムの合計

イベントカウンタ： チャンネル 7 は、イベントの立ち上がりエッジをカウントし、チャンネル 8 はイベントのすべてのエッジをカウントします。どちらのチャンネルも、`P_Ch0708DigOut` パラメータによりコントロールされます。

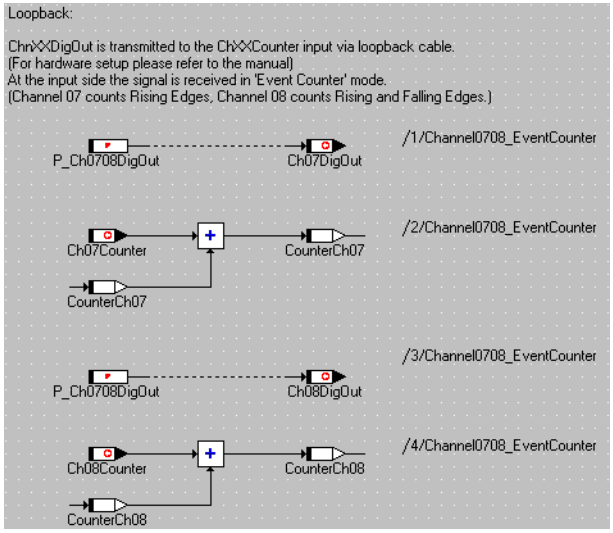


図 11-8 ES1325 ― イベントカウンタ

LED: ここでは LED1 ～ 3 用の出力シグナルがセットされます。LED1 は `log` タイプ、LED2 は `sdisc` タイプ、LED3 は `cont` タイプのパラメータによりそれぞれコントロールされます。

注記

`cont` と `sdisc` の値を使用して LED をコントロールすることは可能ですが、**通常は、お勧めできません。**
適切なモデリングやプログラミングのためには、case 文などを用いて論理変数にマッピングすることをお勧めします。

LED4 は `Ch03DigOut` メッセージによりコントロールされます。

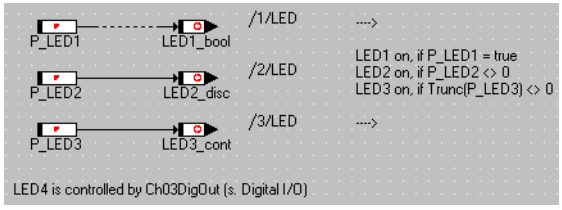
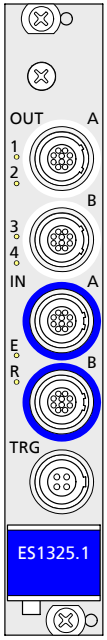


図 11-9 ES1325 ― PWM

11.4.1 ES1325 ボード

下図は、ES1325 ボードのフロントパネルを示しています。



ポート OUT-A と OUT-B にはそれぞれ 8 個の出力チャンネルがまとめられ、IN-A と IN-B にはそれぞれ 8 個の入力チャンネルがまとめられています。また、トリガ入力（2 チャンネル）は、TRG ポートに接続します。LED1 ～ 4 は、モニタ用です。

実験を確実に行えるようにするために、必ず、以下のように正しく配線してください。このチュートリアルでは OUT-A と IN-A、さらに 11.5 項では TRG のポートを使用します。以下の配線で、ES1325 の 2 つのプロジェクト（トリガを使用する／しない）を実行できます。

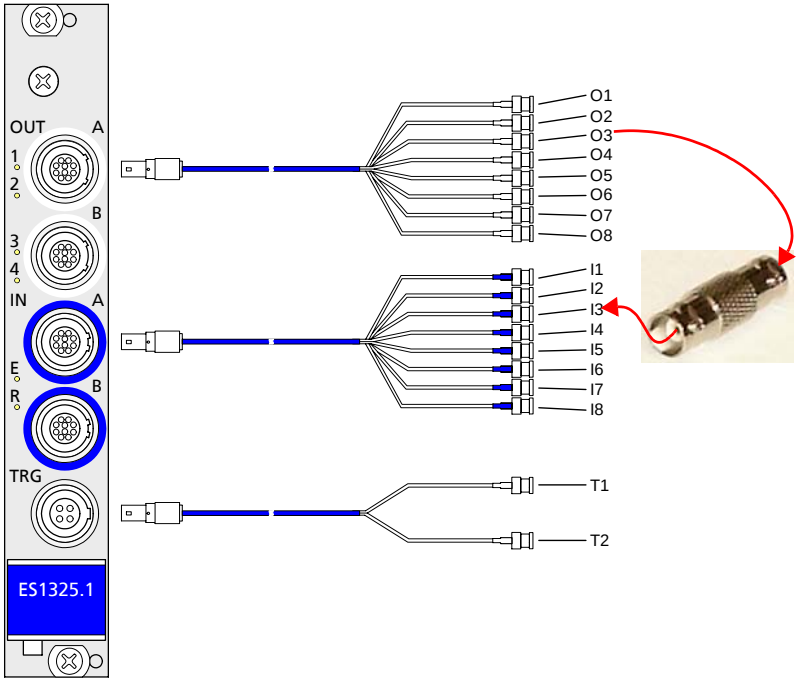


図 11-10 ES1325 の配線

下の表は、どの出力とどの入力を接続するかを示しています。入力 I1 と I2 には何も接続しません。

出力	O1	O2	O3	O4	O5	O6	O7	O8
入力			I3	I4	I5	I6	I7	I8
トリガ	T1	T2						

注記

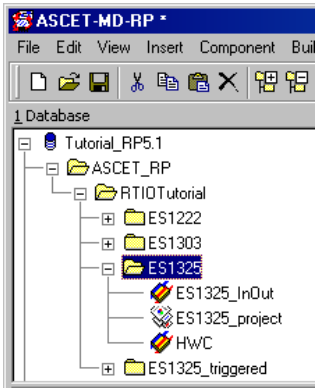
入力と出力のための接続部品は、ユーザー側で用意してください。

1 つの ES1000 システム内で、最大 4 枚の ES1325 ボードを使用できます。

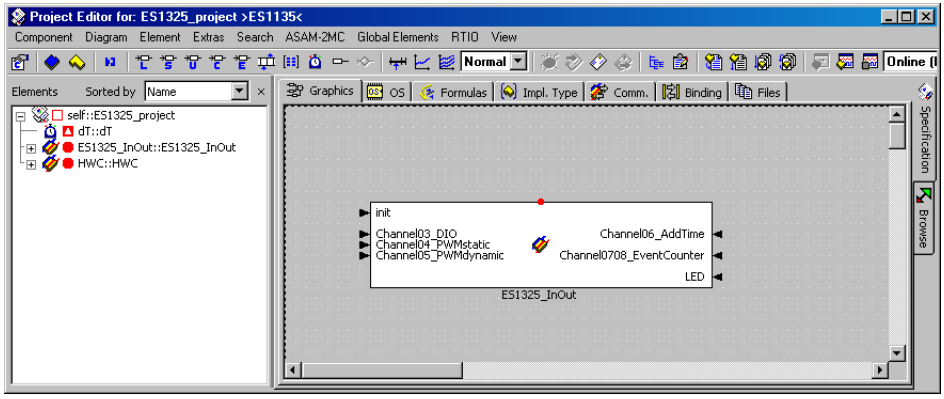
11.4.2 サンプルプロジェクト

サンプルプロジェクトを開く：

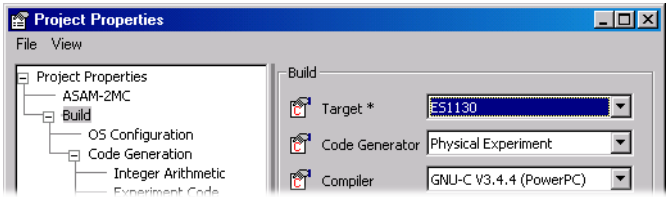
- コンポーネントマネージャで ASCET_RP/RTIOTutorial/ES1325 というフォルダを選択します。



- ES1325_project というプロジェクトを開きます。



- **Project Properties** ボタンをクリックします。
  “Project Properties”ウィンドウが開きます。



- “Build”ノードで以下のオプションを設定します。
  `Target: >ES1130< または >ES1135<`
  `Compiler: GNU-C * (PowerPC)`

注記

RTIO との通信に使用できるのは、“Exported”として宣言されたメッセージのみです。

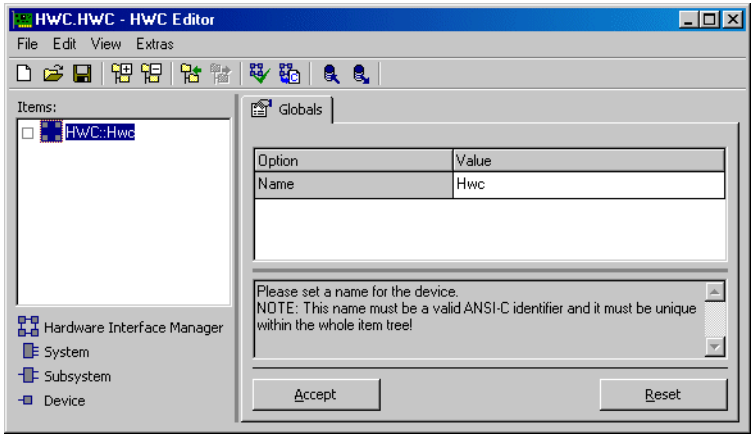
注記

通常、ハードウェアコンフィギュレーションを作成するには、まず HWC というＣコードモジュールを作成し、それをプロジェクトに組み込んでからその内容を編集しますが、このチュートリアルでは、すでにこの HWC モジュールがプロジェクトに組み込まれています。

HWC エディタを開く：

- プロジェクトエディタで **RTIO → Open Editor** を選択します。

  HWC エディタが開きます。



ハードウェアコンフィギュレーション（HWC）を作成する：

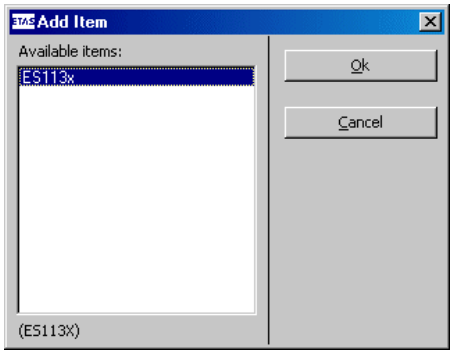
各ハードウェアは、アイテムリスト内にツリー構造の形で定義します。このツリーのルートには、必ず HWC というアイテムが存在します。

- HWC エディタで **Edit → Add Item** を選択します。

または

- **Add Item...** ボタンをクリックします。

  “Add Item” ダイアログボックス が開きます。



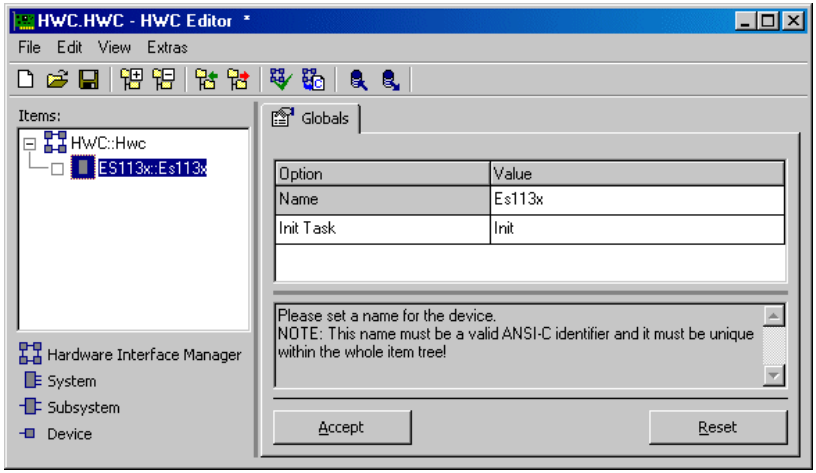
  現在選択されているアイテムの次の階層レベルに追加できるアイテムの一覧が表示されます。

- ES113x を選択します。

  このアイテムは、ES1000.x システムで使用される PowerPC プロセッサボード ES1130 または ES1135 に相当します。

- **OK** をクリックします。

  “Items” リストに ES113x というアイテムが追加されます。

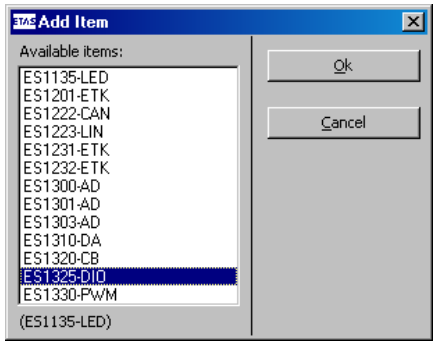


- “Items” リストから ES113x を選択します。

“Globals”タブには、“Init Task”オプションの欄に `Init` というデフォルトのタスク名が表示されます。サンプルプロジェクトの OS エディタで同じ名前の Init タスクが定義されているので、ここで他のタスクを選択する必要はありません。

ES1325 の組み込みと設定を行う：

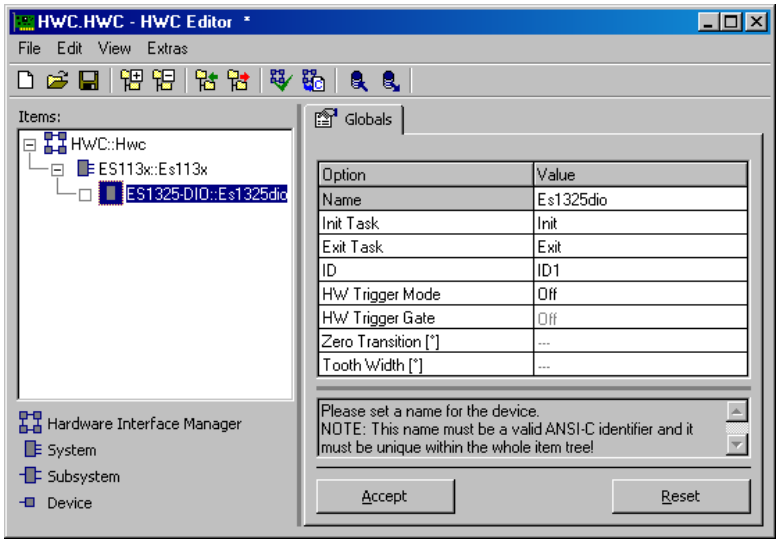
- **Edit → Add Item...** を選択して、次の階層レベルで選択可能なアイテムの一覧を開きます。



- `ES1325-DIO` を選択します。

  このアイテムは、ES1325 インターフェースを定義するために使用されます。

- **OK** をクリックします。
  “Items”リストに ES1325-DIO というアイテムが追加されます。



- “Items”リストから ES1325-DIO を選択します。

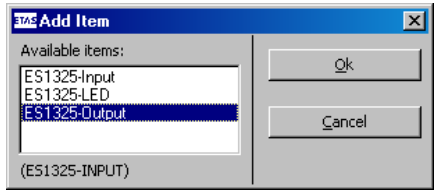
“Globals”タブには、“Init Task”および“Exit Task”オプションの欄に Init および Exit というデフォルトのタスク名が表示されます。サンプルプロジェクトの OS エディタで同じ名前のタスクが定義されているので、ここで他のタスクを選択する必要はありません。

“ID”オプションの番号は、自動的に設定されます。この ES1325-DIO アイテムの他には同じタイプのボードは組み込まれていないため、値は ID1 となります。

- **Accept** ボタンで設定内容を確定します。

デバイスを作成する：

- “Items”リストから ES1325-DIO を選択します。
- **Edit → Add Item...** を選択して、次の階層レベルに組み込むことができるアイテムの一覧を表示します。



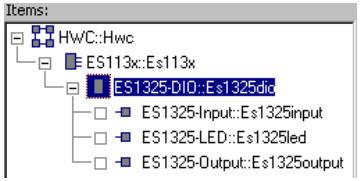
- ES1325-Input を選択して **OK** をクリックします。

  これは、ES1325 の入力チャンネルを定義するためのアイテムです。
- さらに、ES1325-Output と ES1325-LED というデバイスも追加します。

  ES1325-Output は ES1325 の出力チャンネルを定義するためのもので、ES1325-LED は LED を定義するためのものです。

  このサンプルプロジェクトにおいては、3 つのデバイスすべてについて、“Globals” タブではデフォルト設定を使用します。

これで、サンプルシステムのハードウェア構成に対応するアイテムツリーが完成しました。



11.4.4 ES1325 の HWC 設定

次に、各入力と出力チャンネル、および LED について設定します。つまり、出力チャンネルから供給される信号を入力チャンネルで測定し、また内部の処理状態を LED でモニタできるように、ハードウェアコンフィギュレーションのパラメータ設定を行います。

“Globals” タブ： “Globals” タブについては、ユーザーが設定する必要のあるオプションはありません。

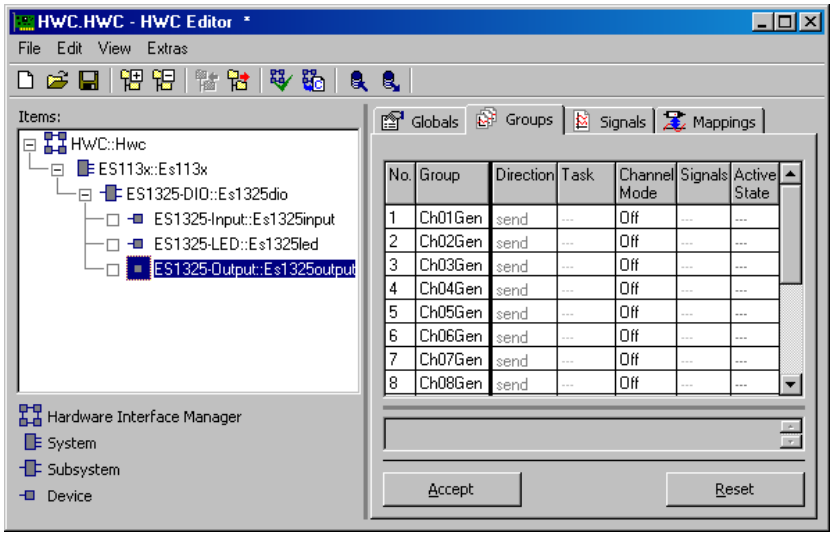
出力 ― ES1325-Output デバイス

“Groups” タブ： “Groups” タブではシグナルグループごとの設定を行います。

ES1325-Output デバイスには、各出力シグナル（Ch<n>Gen、<n> = 01 ～ 16）ごとにシグナルグループが割り当てられています。ここでは、使用される各グループについて、モード、タスク割り当て、生成されるシグナル、トリガタイプ（レベルまたはエッジ）を設定します。

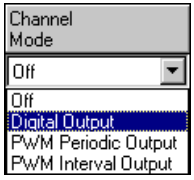
“Groups”タブの設定を行う：

- “Groups”タブを選択します。



シグナルグループ 3 ～ 8 について、以下の設定を行います。

1. チャンネルモードを選択します。



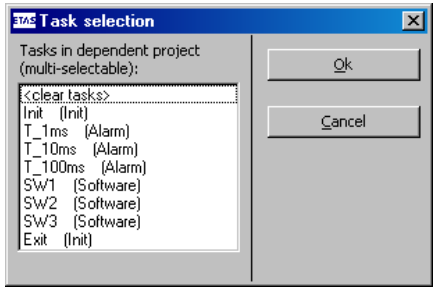
- 各グループについて、“Channel Mode”列で以下のモードを選択します。

Group	Channel Mode
Ch03Gen, Ch07Gen, Ch08Gen	Digital Output
Ch04Gen, Ch05Gen, Ch06Gen	PWM Periodic Output

“Active State”列には、設定されたモードに対応するデフォルト値がセットされます。
“Task”列には、シグナル転送を行うタスクを割り当てることができます。

2. タスクを割り当てます。

- 各行の“Task”列をダブルクリックして“Task selection”ダイアログボックスを開きます。



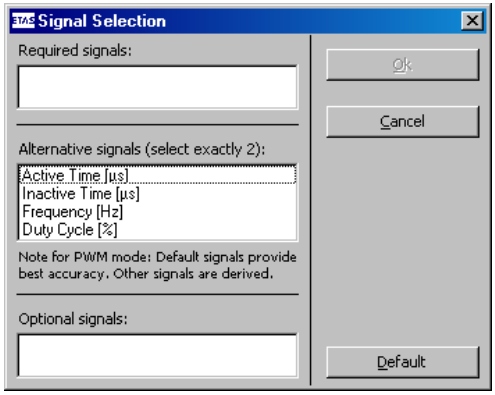
- タスク選択リストから、データ転送用に以下のタスクを選択します。

Group	Task
`Ch03Gen, Ch04Gen`	`T_100ms`
`Ch05Gen, Ch06Gen, Ch07Gen, Ch08Gen`	`T_10ms`

- **OK** をクリックします。
  選択したタスクが“Task”列に表示されます。

3. シグナルを選択します。

   - 各行の“Signals”列をダブルクリックして“Signal Selection”ダイアログボックス を開きます。



   シグナルについての説明は、249 ページの「Signals」という項を参照してください。

   - 以下のシグナルを選択します。

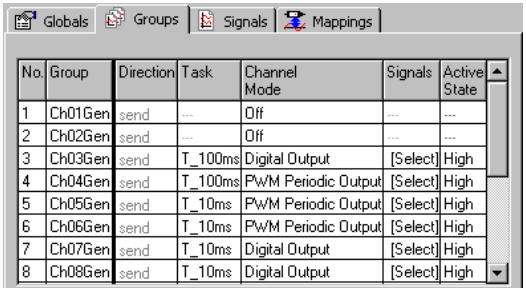
Group	Signals
`Ch03Gen`, `Ch07Gen`, `Ch08Gen`	デフォルト設定
`Ch04Gen`	`Active Time [µs]`, `Inactive Time [µs]`
`Ch05Gen`, `Ch06Gen`	`Frequency [Hz]`, `Duty Cycle [%]`

   - **OK** をクリックして選択を確定します。

4. レベルを選択します。
   すべてのグループがデフォルト設定の `High` を使用するので、“Active State”列については何も変更する必要はありません。

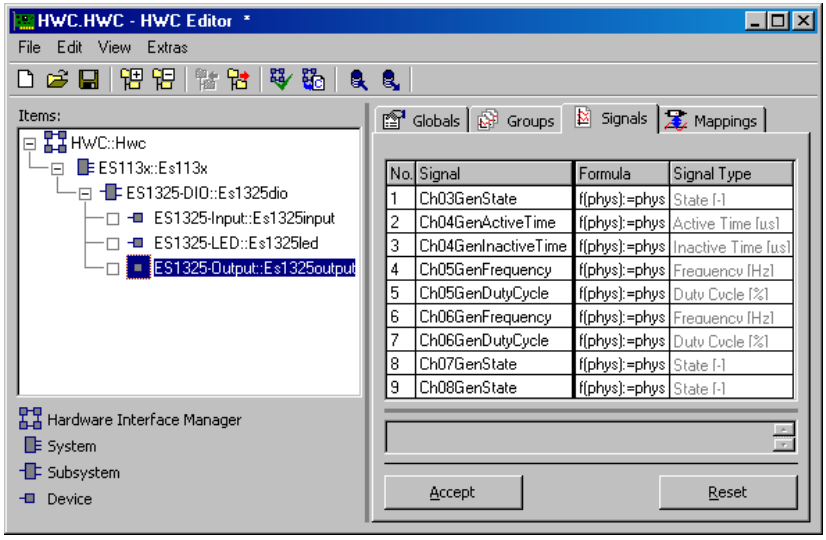
   - すべての設定が終了したら、**Accept** ボタンで設定内容を保存します。

シグナルグループの設定が終わると、“Groups”タブの表示は以下のようになります。



No.	Group	Direction	Task	Channel Mode	Signals	Active State
1	Ch01Gen	send	---	Off	---	---
2	Ch02Gen	send	---	Off	---	---
3	Ch03Gen	send	T_100ms	Digital Output	[Select]	High
4	Ch04Gen	send	T_100ms	PWM Periodic Output	[Select]	High
5	Ch05Gen	send	T_10ms	PWM Periodic Output	[Select]	High
6	Ch06Gen	send	T_10ms	PWM Periodic Output	[Select]	High
7	Ch07Gen	send	T_10ms	Digital Output	[Select]	High
8	Ch08Gen	send	T_10ms	Digital Output	[Select]	High

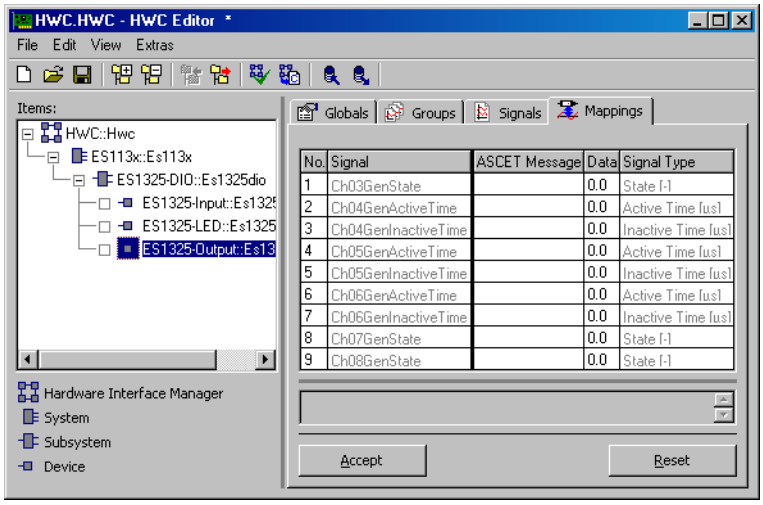
“Signals”タブ： ここには“Groups”タブで定義されたシグナルが表示されます。



No.	Signal	Formula	Signal Type
1	Ch03GenState	f(phys):=phys	State [-]
2	Ch04GenActiveTime	f(phys):=phys	Active Time [us]
3	Ch04GenInactiveTime	f(phys):=phys	Inactive Time [us]
4	Ch05GenFrequency	f(phys):=phys	Frequency [Hz]
5	Ch05GenDutyCycle	f(phys):=phys	Duty Cycle [%]
6	Ch06GenFrequency	f(phys):=phys	Frequency [Hz]
7	Ch06GenDutyCycle	f(phys):=phys	Duty Cycle [%]
8	Ch07GenState	f(phys):=phys	State [-]
9	Ch08GenState	f(phys):=phys	State [-]

このタブではシグナルの名前と式を編集できますが、サンプルプロジェクトにおいては、何も*変更する必要はありません*。

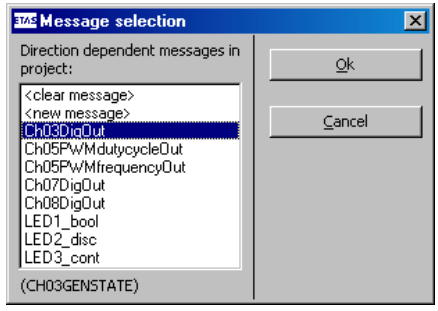
“Mappings” タブ： ここではプロジェクトのシグナルと ASCET メッセージを互いに関連付けます。“ASCET Message” 列のセルをクリックすると選択ダイアログが開きます。そのダイアログには、*Exported* 属性でしかもシグナルグループの転送方向と一致するメッセージ（ここでは *send メッセージ*）だけが表示されます。



シグナルに任意の ASCET メッセージを割り当てる：

- “Mappings” タブを選択します。
- 各行の “ASCET Message” 列をダブルクリックします。

  “Message selection” ダイアログボックスが開きます。



ASCET プロジェクトに含まれるすべての Send メッセージが表示されます。

- 各シグナルについて、以下の ASCET メッセージを選択します。

Signal	ASCET Message
Ch03GenState	Ch03DigOut
Ch05GenFrequency, Ch06GenFrequency	Ch05PWMfrequencyOut
Ch05GenDutyCycle, Ch06GenDutyCycle	Ch05PWMdutycycleOut
Ch07GenState	Ch07DigOut
Ch08GenState	Ch08DigOut

Ch04GenActiveTime と Ch04GenInactiveTime というシグナルには、メッセージを割り当てる代わりに固定値を割り当てます。

- **OK** をクリックします。
- HWC エディタの **Accept** ボタンをクリックして設定内容を保存します。

シグナルに固定値を割り当てる：

ASCET メッセージが割り当てられていないシグナルについて、固定的な値を“Data”列に入力します。

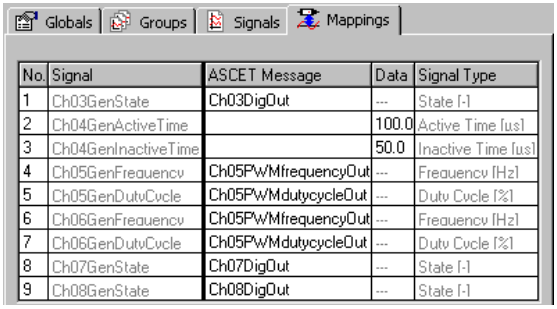
- 各シグナルの“Data”列をダブルクリックします。

そのフィールドが入力ボックスになります。

- 以下の値を入力します。

Signal	Data
Ch04GenActiveTime	100.0
Ch04GenInactiveTime	50.0

すべてのシグナルにメッセージまたは値が割り当てられると、このタブの内容は以下のようになります。



No.	Signal	ASCET Message	Data	Signal Type
1	Ch03GenState	Ch03DigOut	---	State [-]
2	Ch04GenActiveTime		100.0	Active Time [us]
3	Ch04GenInactiveTime		50.0	Inactive Time [us]
4	Ch05GenFrequency	Ch05PWMfrequencyOut	---	Frequency [Hz]
5	Ch05GenDutyCycle	Ch05PWMdutycycleOut	---	Duty Cycle [%]
6	Ch06GenFrequency	Ch05PWMfrequencyOut	---	Frequency [Hz]
7	Ch06GenDutyCycle	Ch05PWMdutycycleOut	---	Duty Cycle [%]
8	Ch07GenState	Ch07DigOut	---	State [-]
9	Ch08GenState	Ch08DigOut	---	State [-]

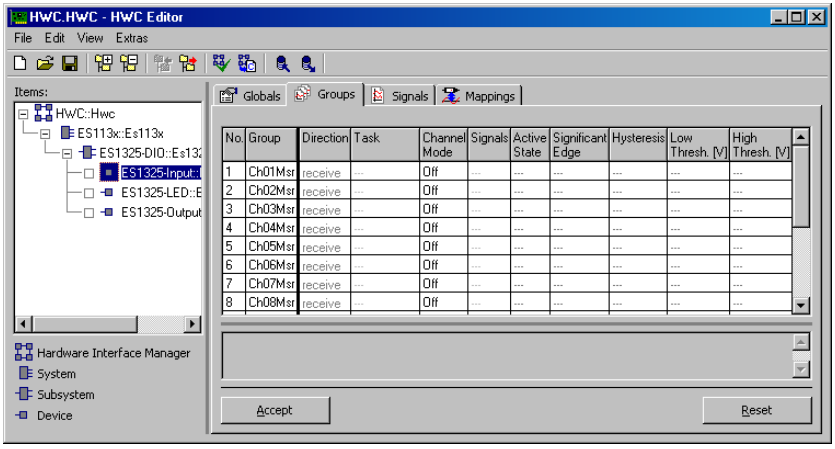
入力 ― ES1325-Input デバイス

“Groups”タブ： “Groups”タブではシグナルグループごとの設定を行います。

ES1325-Input デバイスには、各入力シグナル（Ch<n>Msr、<n> = 01 ～ 16）ごとにシグナルグループが割り当てられています。ここでは、使用される各グループについて、モード、タスク割り当て、生成されるシグナル、トリガタイプ（レベルまたはエッジ）を設定します。

“Groups”タブの設定を行う：

- “Groups”タブを選択します。



シグナルグループ 3 ～ 8 について、以下の設定を行います。

1. チャンネルモードを選択します。

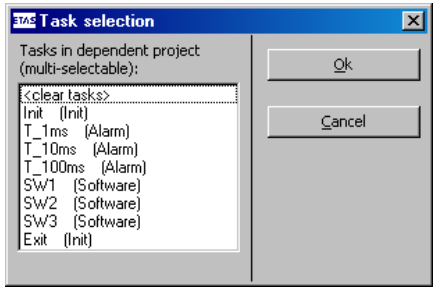
- 各行の“Channel Mode”列で以下のモードを選択します。

Group	Channel Mode
Ch03Msr	Digital Input
Ch04Msr, Ch05Msr	PWM Input
Ch06Msr	Additive Time
Ch07Msr, Ch08Msr	Event Counter

“Active State”、“Significant Edge”、“Hysteresis”、“Low Thresh. [V]”、“High Thresh. [V]”の各列に、各モードのデフォルト値がセットされます。
“Task”列には、シグナル転送を行うタスクを割り当てることができます。

2. タスクを割り当てます。

- 各行の“Task”列をダブルクリックして“Task selection”ダイアログボックスを開きます。



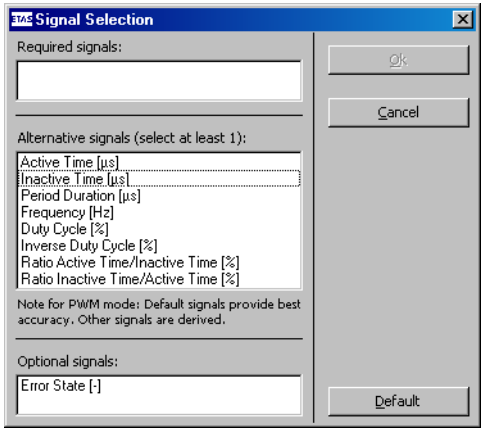
- タスク選択リストから、データ転送用に以下のタスクを選択します。

Group	Task
Ch03Msr, Ch04Msr	T_100ms
Ch05Msr, Ch06Msr, Ch07Msr, Ch08Msr	T_10ms

- **OK** をクリックします。
選択したタスクが“Task”列に表示されます。

3. シグナルを選択します。

- 各行の“Signals”列をダブルクリックして“Signal Selection”ダイアログボックス を開きます。



このダイアログボックス で、当該グループ用に生成されるシグナルを指定します。シグナルについての説明は、242 ページの「Signals」という項を参照してください。

- 以下のシグナルを選択します。

Group	Signals
`Ch04Msr`	`Duty Cycle [%]`
`Ch05Msr`	`Active Time [µs]`, `Inactive Time [µs]`
`Ch03Msr`, `Ch06Msr`, `Ch07Msr`, `Ch08Msr`	デフォルト設定

- **OK** をクリックして選択内容を確定します。

4. レベルを選択します。

- 各行の“Active State”列（249 ページを参照してください）をダブルクリックします。

ドロップダウンリストが開きます。

- アクティブステートを以下のシグナルレベルに設定します。

Group	Active State
Ch03Msr	Low
Ch04Msr – Ch08Msr	High

選択した値が“Active State”フィールドに表示されます。

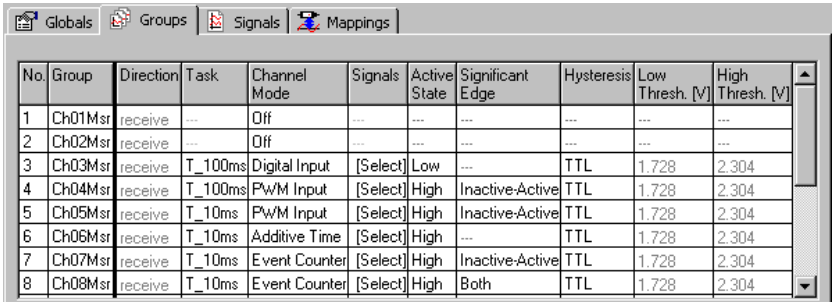
5. エッジを選択します（PWM Input モードと Event Counter モードのシグナルグループのみ）。

- 各行の“Significant Edge”列（243 ページを参照してください）をダブルクリックして、イベントとして使用するエッジを選択します。

Group	Significant Edge
Ch04Msr, Ch05Msr, Ch07Msr	Inactive-Active
Ch08Msr	Both

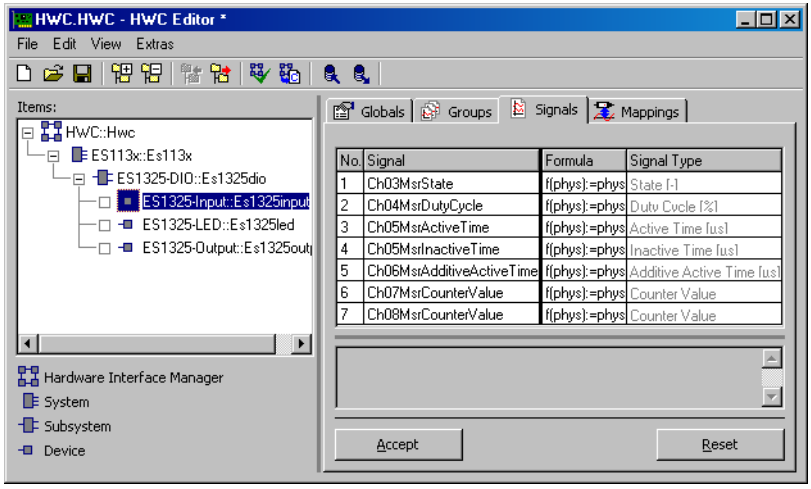
- 必要な設定をすべて終えたら、HWC エディタの **Accept** をクリックして設定を保存します。

シグナルグループの設定が終わると、“Groups”タブの内容は以下のようになります。



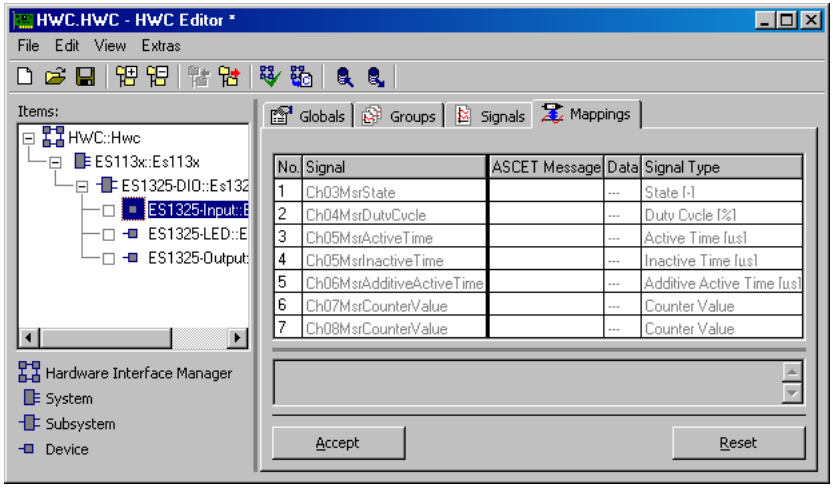
No.	Group	Direction	Task	Channel Mode	Signals	Active State	Significant Edge	Hysteresis	Low Thresh. [V]	High Thresh. [V]
1	Ch01Msr	receive	---	Off	---	---	---	---	---	---
2	Ch02Msr	receive	---	Off	---	---	---	---	---	---
3	Ch03Msr	receive	T_100ms	Digital Input	[Select]	Low	---	TTL	1.728	2.304
4	Ch04Msr	receive	T_100ms	PWM Input	[Select]	High	Inactive-Active	TTL	1.728	2.304
5	Ch05Msr	receive	T_10ms	PWM Input	[Select]	High	Inactive-Active	TTL	1.728	2.304
6	Ch06Msr	receive	T_10ms	Additive Time	[Select]	High	---	TTL	1.728	2.304
7	Ch07Msr	receive	T_10ms	Event Counter	[Select]	High	Inactive-Active	TTL	1.728	2.304
8	Ch08Msr	receive	T_10ms	Event Counter	[Select]	High	Both	TTL	1.728	2.304

“Signals” タブ： ここには“Groups”タブで定義されたシグナルが表示されます。



このタブではシグナルの名前と式を編集できますが、サンプルプロジェクトにおいては、何も*変更する必要はありません*。

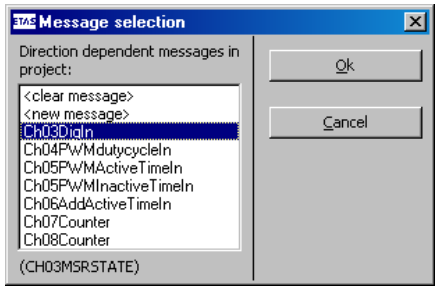
“Mappings” タブ： ここで、プロジェクトのシグナルと ASCET メッセージを互いに関連付けます。“ASCET Message”列のセルをクリックすると選択ダイアログが開きます。そのダイアログには、*Exported* 属性でしかもシグナルグループの転送方向と一致するメッセージ（ここでは *receive メッセージ*）だけが表示されます。



シグナルに任意の ASCET メッセージを割り当てる：

- “Mappings”タブを選択します。
- 各行の“ASCET Message”列をダブルクリックします。

  “Message selection”ダイアログボックス が開きます。



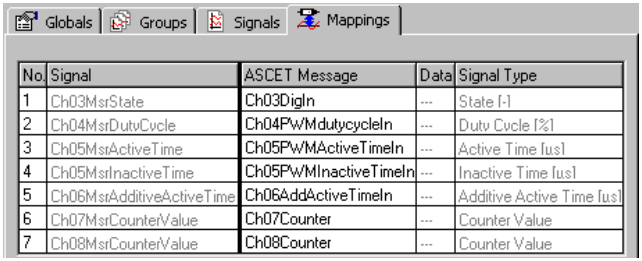
  ASCET プロジェクトに含まれるすべての Receive メッセージが表示されます。

- 各シグナルについて、以下の ASCET メッセージを選択します。

Signal	ASCET Message
Ch03MsrState	Ch03DigIn
Ch04MsrDutyCycle	Ch04PWMdutycycleIn
Ch05MsrActiveTime	Ch05PWMActiveTimeIn
Ch05MsrInactiveTime	Ch05PWMInactiveTimeIn
Ch06MsrAdditiveActiveTime	Ch06AddActiveTimeIn
Ch07MsrCounterValue	Ch07Counter
Ch08MsrCounterValue	Ch08Counter

- **OK** をクリックします。
- HWC エディタの **Accept** ボタンをクリックして設定内容を保存します。

すべてのシグナルにメッセージが割り当てられると、このタブの表示は以下のようになります。



No.	Signal	ASCET Message	Data	Signal Type
1	Ch03MsrState	Ch03DigIn	---	State [-]
2	Ch04MsrDutyCycle	Ch04PWMdutycycleIn	---	Duty Cycle [%]
3	Ch05MsrActiveTime	Ch05PWMActiveTimeIn	---	Active Time [us]
4	Ch05MsrInactiveTime	Ch05PWMInactiveTimeIn	---	Inactive Time [us]
5	Ch06MsrAdditiveActiveTime	Ch06AddActiveTimeIn	---	Additive Active Time [us]
6	Ch07MsrCounterValue	Ch07Counter	---	Counter Value
7	Ch08MsrCounterValue	Ch08Counter	---	Counter Value

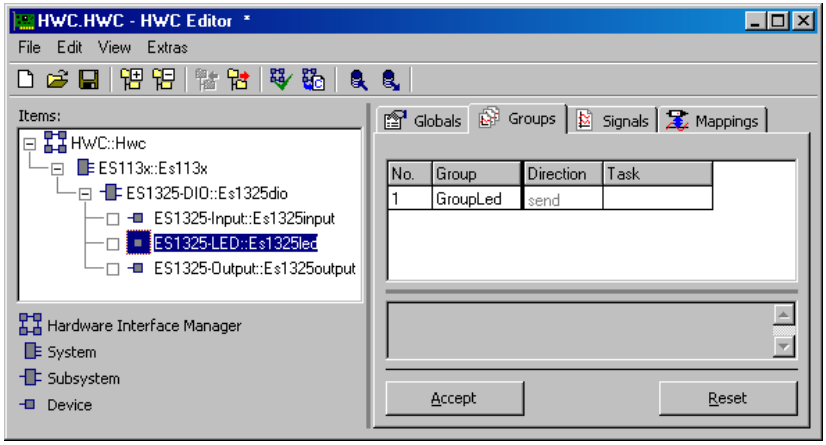
LED ― ES1325-LED *デバイス*

“Groups”タブ： “Groups”タブではシグナルグループごとの設定を行います。

ES1325-LED デバイスには、1 つのシグナルグループ（GroupLED）が割り当てられています。ここでは、このグループにタスクを割り当てます。

“Groups”タブの設定を行う：

- “Groups”タブを選択します。

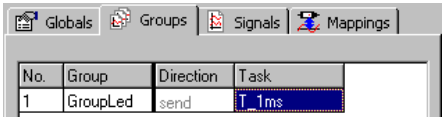


- 各行の“Task”列をダブルクリックして“Task selection”ダイアログボックスを開きます。
- タスク選択リストから、T_1ms (Alarm) タスクを選択します。
- **OK** をクリックします。

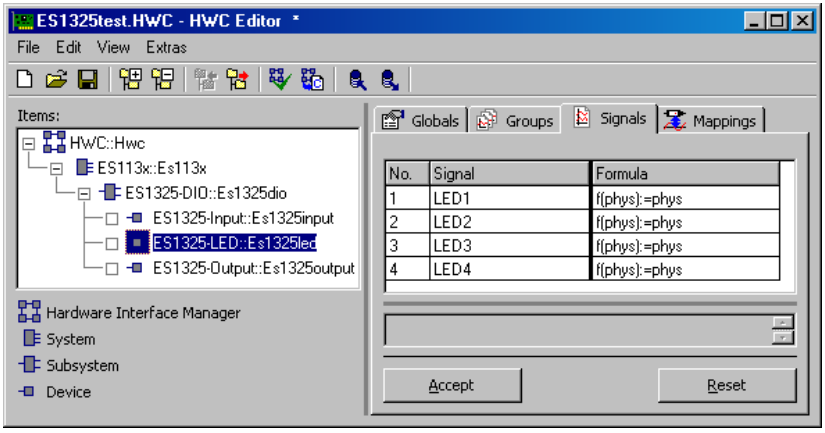
  選択したタスクが“Task”列に表示されます。

- HWC エディタの **Accept** をクリックして設定を保存します。

“Groups”タブの表示は以下のようになります。

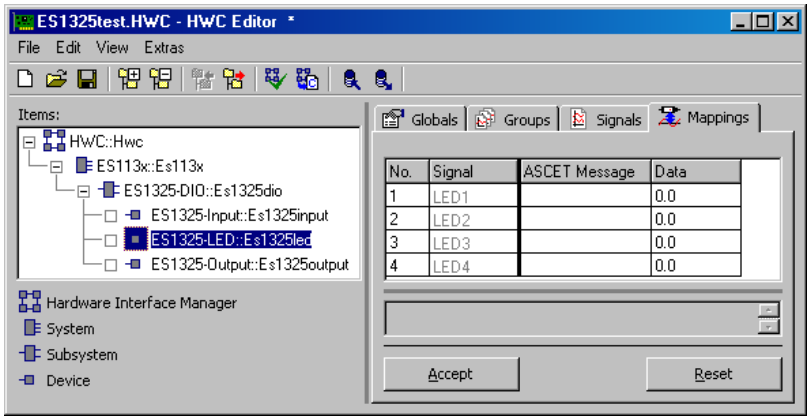


“Signals”タブ： ここにはシグナルグループ `GroupLED` に属するシグナルが表示されます。ボード上の各 LED につき 1 つのシグナルが定義されています。



このタブではシグナルの名前と式を編集できますが、サンプルプロジェクトにおいては、何も*変更する必要はありません*。

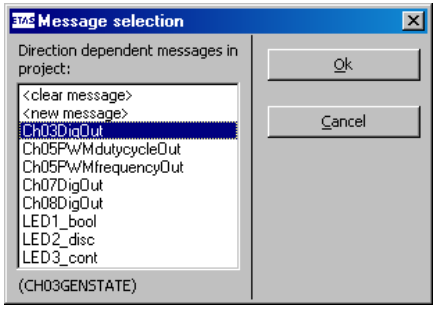
“Mappings” タブ： ここではプロジェクトのシグナルと ASCET メッセージを互いに関連付けます。“ASCET Message” 列のセルをクリックすると選択ダイアログが開きます。そのダイアログには、*Exported* 属性でしかもシグナルグループの転送方向と一致するメッセージ（ここでは *send メッセージ*）だけが表示されます。



シグナルに任意の ASCET メッセージを割り当てる：

- “Mappings” タブを選択します。
- 各行の “ASCET Message” 列をダブルクリックします。

  “Message selection” ダイアログボックス が開きます。



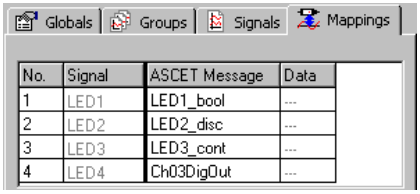
ASCET プロジェクトに含まれるすべての Send メッセージが表示されます。

- 各シグナルについて、以下の ASCET メッセージを選択します。

Signal	ASCET Message
LED1	LED1_bool
LED2	LED2_disc
LED3	LED3_cont
LED4	Ch03DigOut

- **OK** をクリックします。
- HWC エディタの **Accept** ボタンをクリックして設定内容を保存します。

“Mappings”タブの内容は以下のようになります。



11.4.5 ハードウェアコンフィギュレーションの保存

作成したハードウェアコンフィギュレーションは、プロジェクトのファイルコンテナ内に保存したり、または DOS ファイル（*.HWX）として保存することができます。293 ページの 11.2.5 項を参照してください。

注記

サンプルプロジェクトの参考用コンフィギュレーションは、ES1325.hwk という名前で保存されています。このファイルを上書きしないように注意してください。

11.4.6 HWC モジュール用のコードの生成

続いて、HWC モジュールのコード生成を行います。コード生成の方法は、294 ページの 11.2.6 項を参照してください。

11.4.7 サンプルプロジェクトの実験

前項までで RTIO に関する設定がすべて完了したので、HWC エディタを閉じ、次にプロジェクトエディタで実験ターゲット用の標準的なコードを生成します。

注記

HWC モジュールをプロジェクトエディタのグラフィック表示に含めることはお勧めできません。HWC モジュールは RTIO のコード生成が行われるたびに変わるため、予期しない形で表示されてしまうためです。

実験ターゲット用のコードを生成する：

- プロジェクトエディタから **Component → Build** を選択して、プロジェクト全体のコードを生成します。
- **Component → View Generated Code** を選択して生成されたコードを表示し、内容を確認します。

オンライン実験を行う：

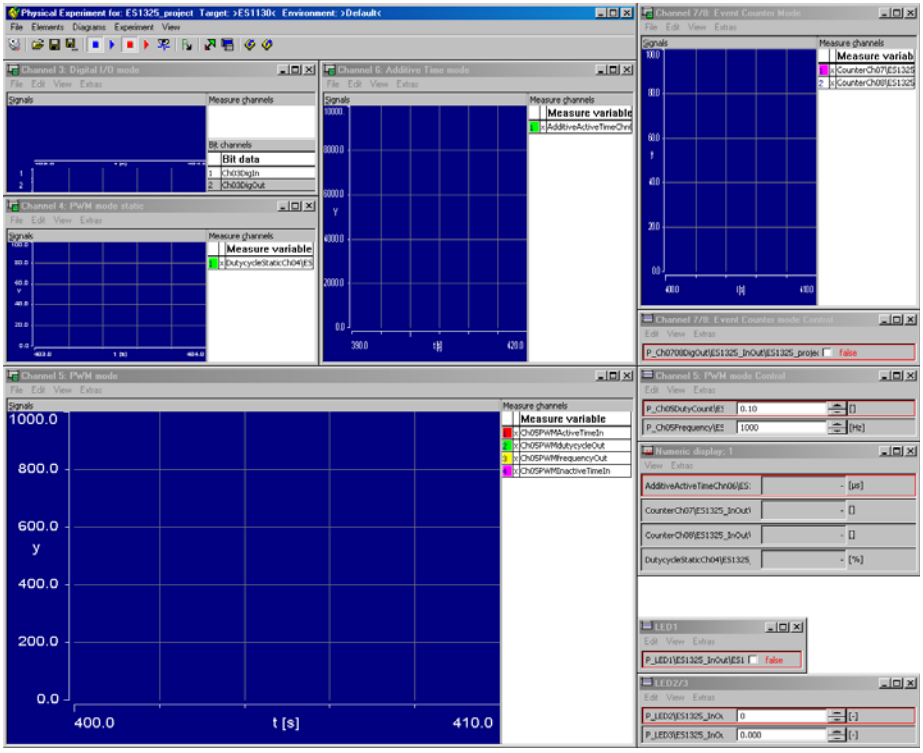
- ES1325 の入力ポートと出力ポートがまだ接続されていない場合は 317 ページの「接続」の項の記述に従って接続し、ES1000.x の電源をオンにします。

- プロジェクトエディタの“Experiment Target”コンボボックスから `Online (RP)` を選択します。

  `Offline (RP)` は、ターゲット上でオフライン実験を行うための設定です。

- **Component → Open Experiment** を選択します。

  “Physical Experiment“ウィンドウが開き、あらかじめ定義されている実験環境（5 つのオシロスコープ、1 つの数値ディスプレイ、4 つの適合エディタ で構成されます）が開きます。



- “Physical Experiment”ウィンドウ のメニューから **Experiment → Start ERCOS** を選択します。

または

- **Start ERCOS** ボタンをクリックします。

  オペレーティングシステムが起動し、モデルが実行されます。

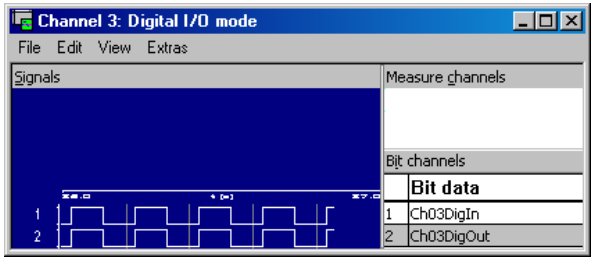
- “Physical Experiment”ウィンドウ のメニューから **Experiment → Start Measurement** を選択します。

または

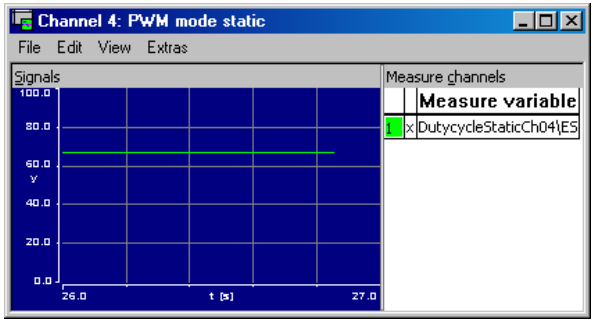
- **Start Measurement** ボタンをクリックします。
  ASCET メッセージの値が、オシロスコープと数値ディスプレイに表示されます。

各モデルブロックについて、以下のような測定や適合を行えます。

デジタル I/O: “Channel 3: Digital I/O”というタイトルのオシロスコープには、`Ch03DigIn` と `Ch03DigOut` というメッセージの内容が表示されます。

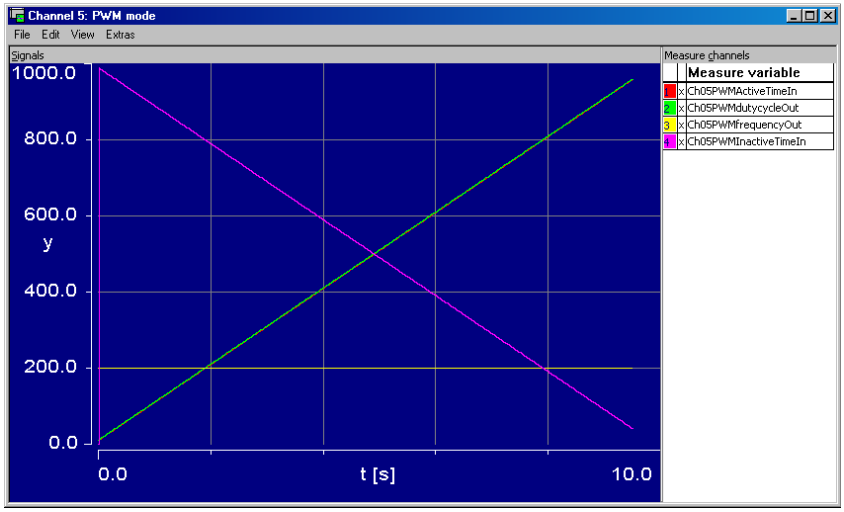


PWM: : “Channel 4: PWM mode static”というタイトルのオシロスコープには、`DutycycleStaticCh04` という変数の内容が表示されます。この変数は PWM シグナルのデューティサイクルを記録するものです。アクティブ時間と非アクティブ時間にはどちらも固定的な値が定義されているので、この変数の値は一定です。



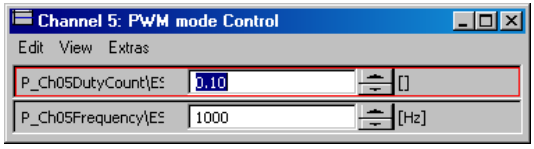
“Channel 5: PWM mode”というタイトルのオシロスコープには、`Ch05PWMActiveTimeIn`（赤いライン）と `Ch05PWMInactiveTimeIn`（紫のライン）という入力メッセージの内容が表示されます。これらのメッセージには、PWM シグナルのアクティブ時間と非アクティブ時間が格納されています。また、`Ch05PWMdutycycleOut`（緑のライン）と `Ch05PWMfrequencyOut`（黄色い

ライン）という出力メッセージも表示されます。これらのメッセージには、PWM シグナルの周波数（Hz）とデューティーサイクル（%）が格納されています。実験が開始されると、`Ch05PWMActiveTimeIn` の Y 軸が表示されます。

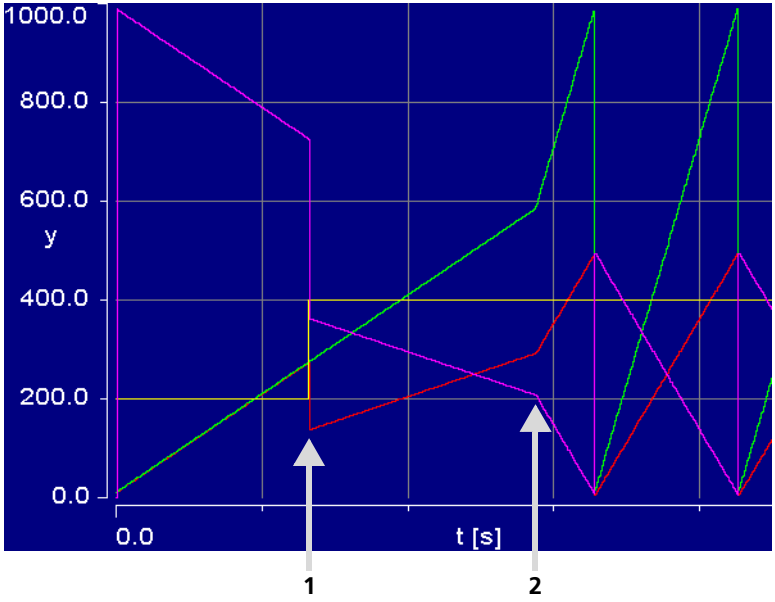


“Channel 5: PWM mode Control” というタイトルの適合エディタ に表示されているパラメータ `P_Ch05Frequency` の値を変更すると、周波数が変わり、アクティブ時間と非アクティブ時間も変わります。デューティサイクルは変わりません。

また、同じ適合エディタに表示されているもう 1 つのパラメータ `P_Ch05DutyCount` の値を変更すると、デューティーサイクルの動向が変わり、アクティブ時間と非アクティブ時間も変わります。周波数は変わりません。

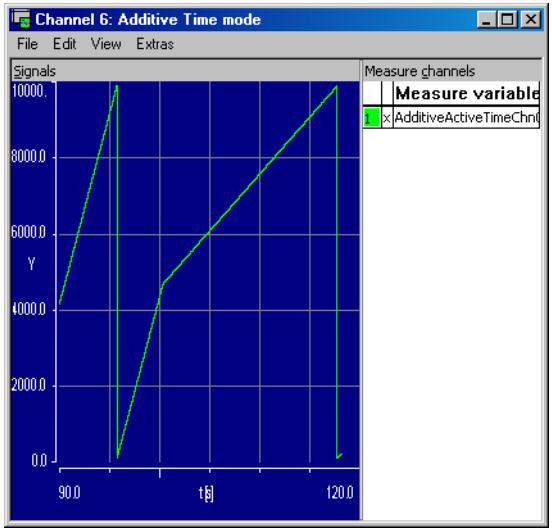


下図の例では、ポイント 1 において周波数が 1000Hz から 2000Hz に変更されたため、`Ch05PWMActiveTimeIn`、`Ch05PWMInactiveTimeIn`、および `Ch05PWMfrequencyOut` のラインが不連続性を示しています。`P_Ch05DutyCount` は変更されなかったので、`Ch05PWMdutycycleOut` の傾斜はそのままです。

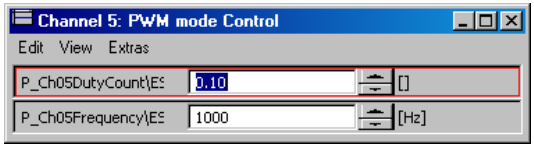


またポイント 2 では、`P_Ch05DutyCount` パラメータの値が `0.1` から `0.5` に変更されたため、`Ch05PWMdutycycleOut`、`Ch05PWMActiveTimeIn`、および `Ch05PWMInactiveTimeIn` のラインの動向が変化しています。このポイントでは周波数は変わっていません。

アクティブタイムの合計： “Channel 6: Additive Time mode”というタイトルのオシロスコープには、シグナルのアクティブ時間の合計を受け取る `AdditiveActiveTimeChn06` という変数が表示されます。

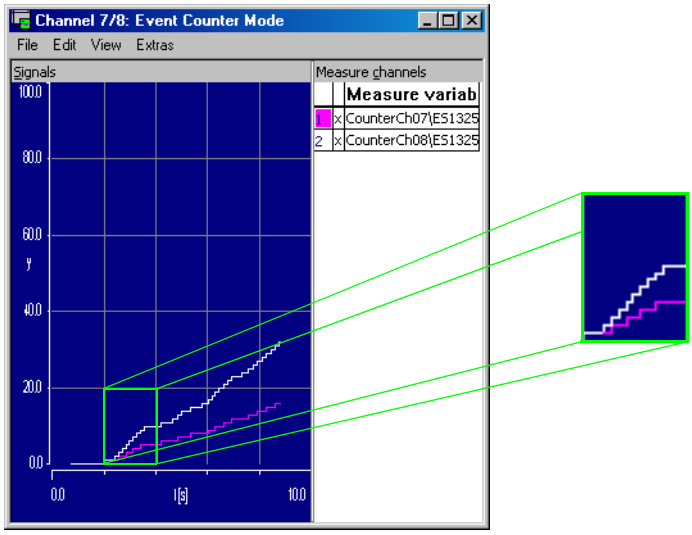


適合エディタ“Channel 5: PWM mode Control”に表示されている `P_Ch05DutyCount` パラメータの値を変更すると、この鋸波の傾斜を変えることができます。上図のラインの最初の部分では `0.1` という値が設定されていますが、次の部分では `0.01` という設定に変更されています。

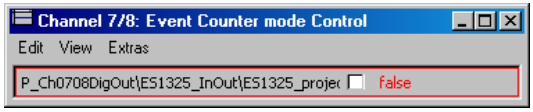


アクティブフェーズの合計時間は周波数には依存しないので、周波数を変更しても `AdditiveActiveTimeChn06` シグナルの値は影響を受けません。

イベントカウンタ： “Channel 7/8: Event Counter mode”というタイトルのオシロスコープには、シグナルカウントを受け取る 2 つの変数、`CounterCh07`（下側のライン）および `CounterCh08`（上側のライン）が表示されます。

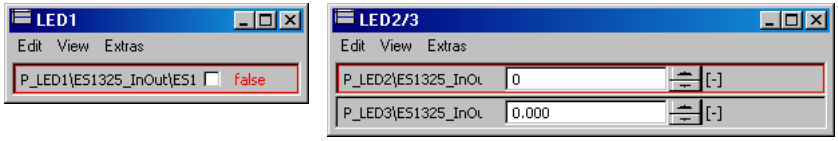


“Channel 7/8: Event Counter mode Control”というタイトルの適合エディタ の `P_Ch0708DigOut` パラメータに `true` と `false` を交互に設定することにより、両方の変数をインクリメントすることができます。



チャンネル 8 は両方向のステート変化をカウントするのに対し、チャンネル 7 は `false` から `true` への変化だけをカウントするので、`CounterCh07` の値は `CounterCh08` の半分の速さで増加します。

LED: “LED1”、および “LED2/3”という適合エディタ に表示されるパラメータ `P_LED1`、`P_LED2`、`P_LED3` を使用して、ES1325 のフロントパネルにある LED をコントロールします。



LED1 は `P_LED1` が `true` になると点灯し、LED2 は `P_LED2` が `0` でない時に点灯し、LED3 は `P_LED3` の値の小数部を切り捨てた結果が 0 でない時に点灯します。

11.5 チュートリアル - ES1325（トリガ使用）

トリガを使用する ES1325 のサンプルプロジェクトには、動的な PWM シグナルを用いるモデル（**PWM** ブロック、図 11-12 を参照してください）が含まれています。このシグナルのデューティーサイクルは、鋸歯シグナルによって決まります。

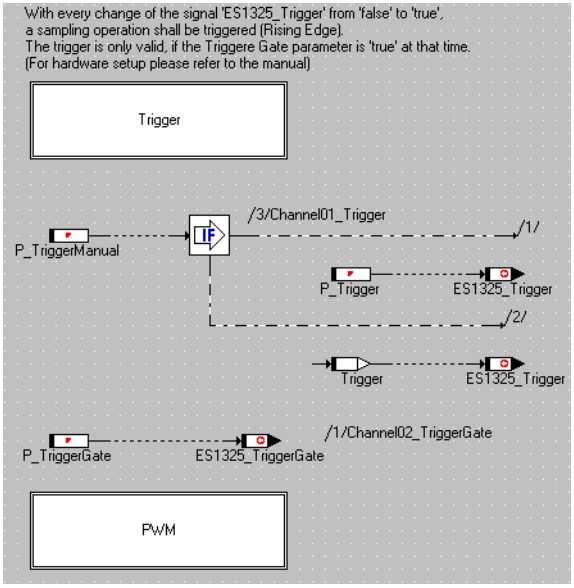


図 11-11 ES1325（トリガあり）― モデル概観

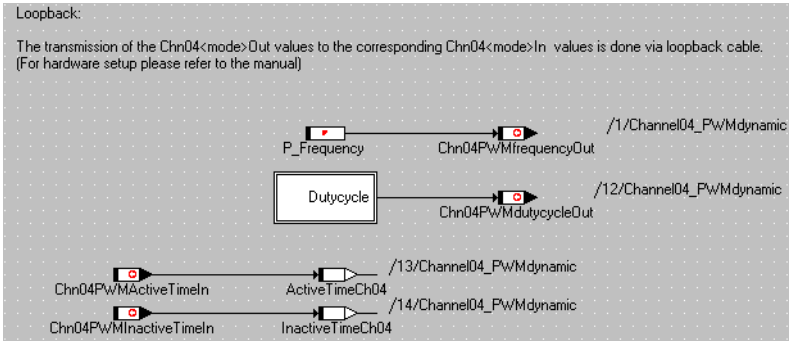


図 11-12 ES1325(トリガあり）― PWM

出力シグナル（Chn04PWMfrequencyOut と Chn04PWMdutycycleOut メッセージ）は連続的に送信されますが、入力シグナル（Chn04PWMActiveTimeIn と Chn04PWMInactiveTimeIn メッセージ）の読み取りは、ハードウェアトリガに
よりコントロールされます。ドリガの発行は、任意に行う（P_TriggerManual パラメータを使用）ことも、自動的にも行うこともできます。

自動的にトリガをかける場合は、Trigger ブロック（図 11-13）内で第 1 のトリガシグナル ES1325_Trigger が計算されます。また任意にトリガをかける場合は、トリガシグナルは P_Trigger パラメータによりコントロールされます。

第 2 のトリガシグナル ES1325_TriggerGate は、常に P_TriggerGate パラメータにより決まります。

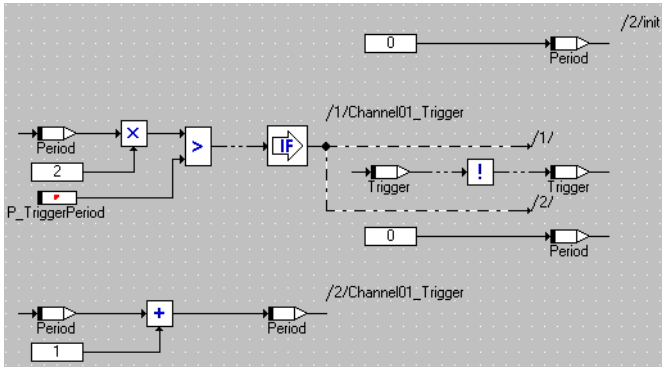


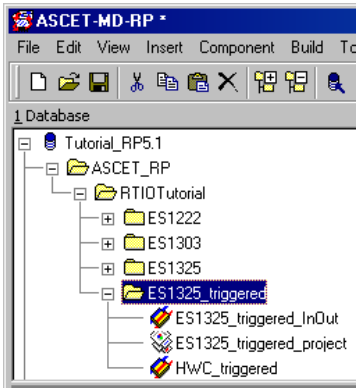
図 11-13 ES1325 ― トリガ

11.5.1 ES1325 ボード

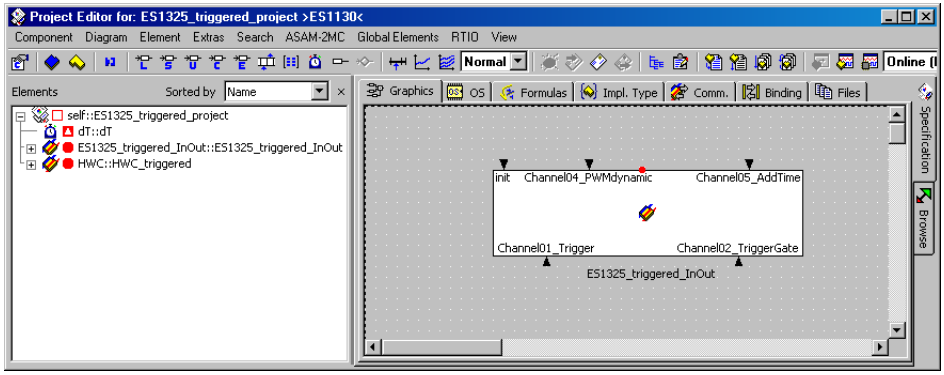
このボードの機能についての概要、および配線については、316 ページの「11.4.1　ES1325 ボード」を参照してください。

サンプルプロジェクトを開く：

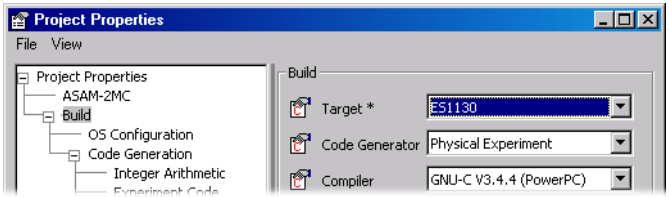
- コンポーネントマネージャで、ASCET_RP/RTIOTutorial/ES1325_triggered というフォルダを選択します。



- ES1325_triggered_project というプロジェクトを開きます。



- **Project Properties** ボタンをクリックします。
  “Project Properties”ウィンドウが開きます。



- “Build”ノードで以下のオプションを設定します。
  `Target: >ES1130< または >ES1135<`
  `Compiler: GNU-C * (PowerPC)`

注記

RTIO との通信に使用できるのは、“Exported”として宣言されたメッセージのみです。

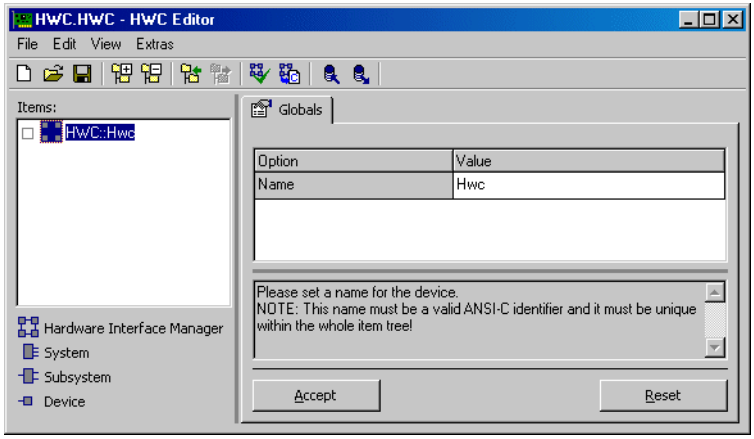
注記

通常、ハードウェアコンフィギュレーションを作成するには、まず HWC という C コードモジュールを作成し、それをプロジェクトに組み込んでからその内容を編集しますが、このチュートリアルでは、すでにこの HWC モジュールがプロジェクトに組み込まれています。

HWC エディタを開く：

- プロジェクトエディタの **RTIO → Open Editor** を選択します。

  HWC エディタが開きます。



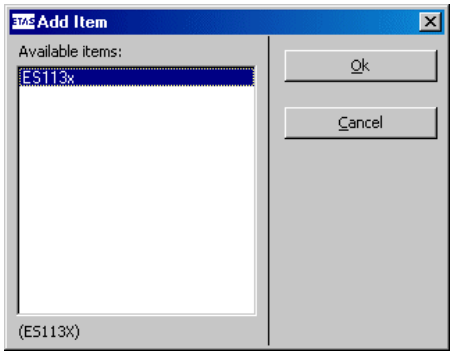
ハードウェアコンフィギュレーション（HWC）を作成する：

各ハードウェアは、アイテムリスト内にツリー構造の形で定義します。このツリーのルートには、必ず HWC というアイテムが存在します。

- HWC エディタで **Edit → Add Item** を選択します。

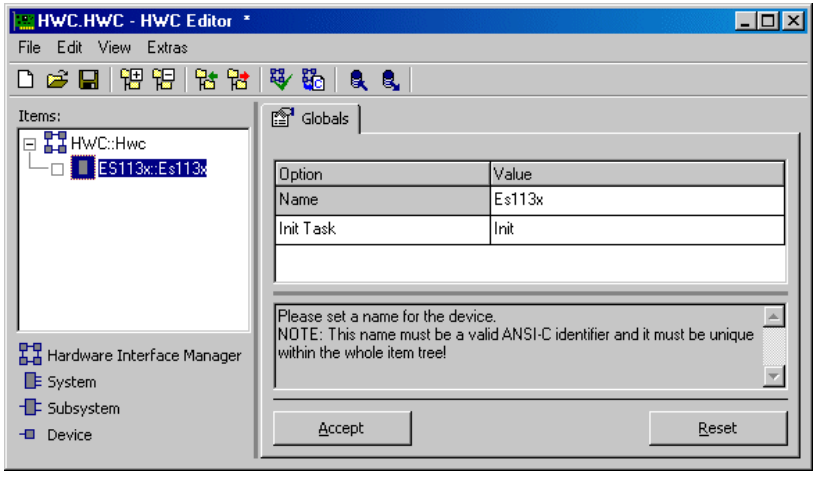
または

- **Add Item...** ボタンをクリックします。
  “Add Item” ダイアログボックス が開きます。



Add Item には、次の階層レベルから使用可能なア現在選択されているアイテムの次の階層レベルに追加できるアイテムの一覧が表示されます。

- ES113x を選択します。
  このアイテムは、ES1000.x システムで使用される PowerPC プロセッサボード ES1130 または ES1135 に相当します。
- **OK** をクリックします。
  “Items” リストに **ES113x** というアイテムが追加されます。

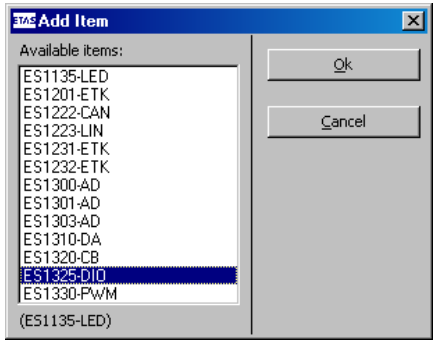


- “Items” リストから ES113x を選択します。

“Globals”タブには、“Init Task”オプションの欄に `Init` というデフォルトのタスク名が表示されます。サンプルプロジェクトの OS エディタで同じ名前の Init タスクが定義されているので、ここで他のタスクを選択する必要はありません。

ES1325 の組み込みと設定を行う：

- **Edit → Add Item...** を選択して、次の階層レベルで選択可能なアイテムの一覧を開きます。

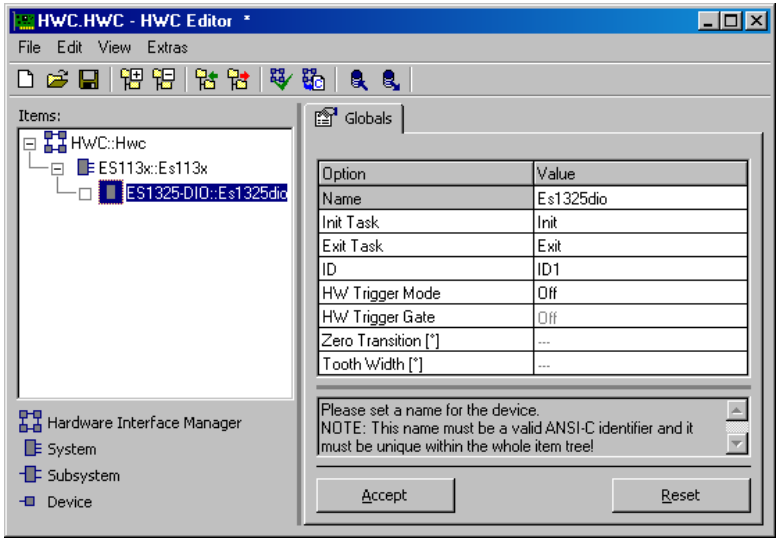


- `ES1325-DIO` を選択します。

  これは ES1325 インターフェースを定義するために使用されます。

- **OK** をクリックします。

  “Items”リストに ES1325-DIO というアイテムが追加されます。

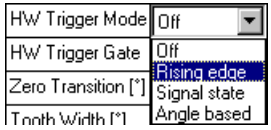


- “Items”リストから ES1325-DIO を選択します。

Init Task と Exit Task というオプションには、322 ページの「ES1325 の組み込みと設定を行う：」の場合と同様に、それぞれ Init と Exit というタスク名を使用できます。

“ID”オプションの番号は、自動的に設定されます。この ES1325-DIO アイテムの他には同じタイプのボードは組み込まれていないため、値は ID1 となります。

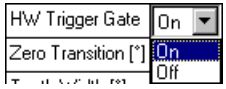
この例題ではハードウェアトリガを使用するので、以下のようにトリガを設定します。



- “HW Trigger Mode”オプションの“Value”列のセルをダブルクリックします（233 ページを参照してください）。

  ドロップダウンリストが開きます。
- Rising edge を選択します。
- “HW Trigger Gate”オプションの“Value”列のセルをダブルクリックします（235 ページを参照してください）。

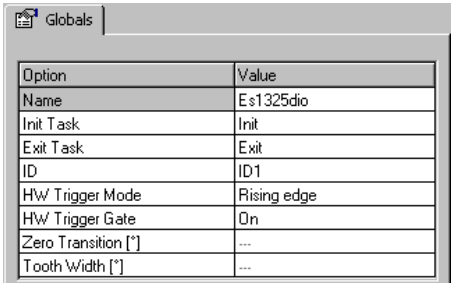
  ドロップダウンリストが開きます。



- On を選択します。

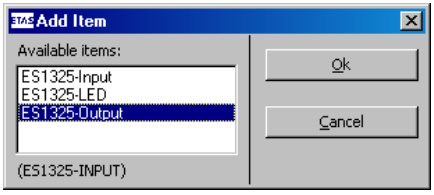
  これらの設定により、使用するハードウェアトリガのタイプが決まります。
- **Accept** を使用してこの設定を有効にします。

このタブの表示は下図のようになります。



デバイスを作成する：

- “Items”リストから `ES1325-DIO` を選択します。
- **Edit → Add Item...** を選択して、次の階層レベルに組み込むことができるアイテムの一覧を表示します。

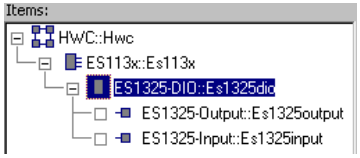


- `ES1325-Output` を選択して **OK** をクリックします。

  これは、ES1325 の出力チャンネルを定義するためのアイテムです。
- さらに、`ES1325-Input` というデバイスも追加します。

  これは ES1325 の入力チャンネルを定義するためのアイテムです。

これで、サンプルシステムのハードウェア構成に対応するアイテムツリーが完全しました。



11.5.4 ES1325 の HWC 設定

次に、各入力と出力チャンネルについて設定します。ここでは、出力チャンネルから供給される信号を入力チャンネルで測定できるようにします。

出力 – ES1325-Output デバイス

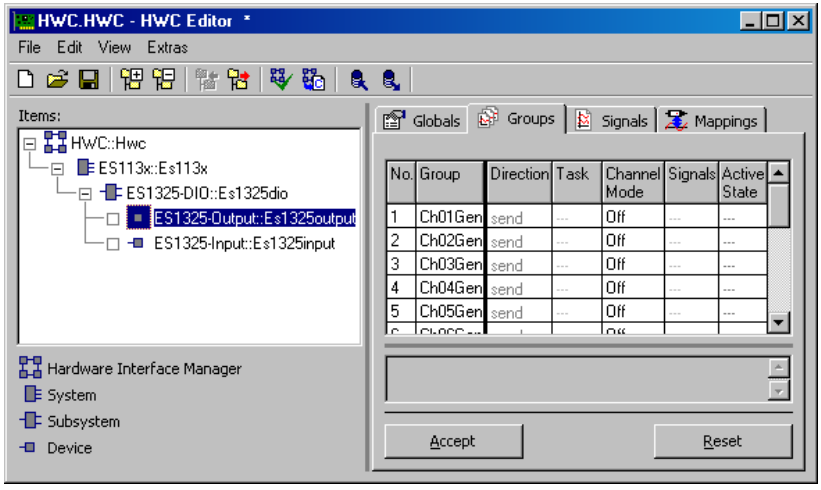
“Globals”タブ： “Globals”タブについては、ユーザーが設定する必要のあるオプションはありません。

“Groups”タブ： “Groups”ではシグナルグループごとの設定を行います。

`ES1325-Output` デバイスには、各出力シグナル（`Ch<n>Gen`、`<n>` = `01` ～ `16`）ごとにシグナルグループが割り当てられています。ここでは、使用される各グループについて、モード、タスク割り当て、生成されるシグナル、トリガタイプ（レベルまたはエッジ）を設定します。

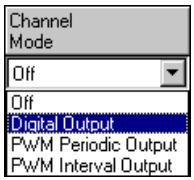
“Groups”タブの設定を行う：

- “Groups”タブを選択します。



シグナルグループ 1、2、4 について、以下の手順を実行します。

1. チャンネルモードを選択します。



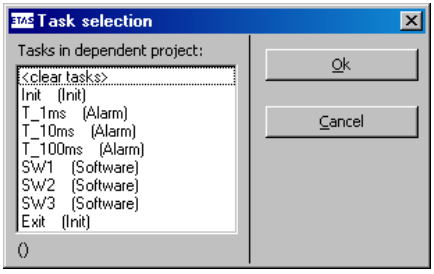
- 各グループについて、“Channel Mode”列で以下のモードを選択します。

Group	Channel Mode
`Ch01Gen, Ch02Gen`	`Digital Output`
`Ch04Gen`	`PWM Periodic Output`

“Active State”列には、設定されたモードに対応するデフォルト値がセットされます。
“Task”列には、シグナル転送を行うタスクを割り当てることができます。

2. タスクを割り当てます。

- 各行の“Task”列をダブルクリックして“Task selection”ダイアログボックスを開きます。



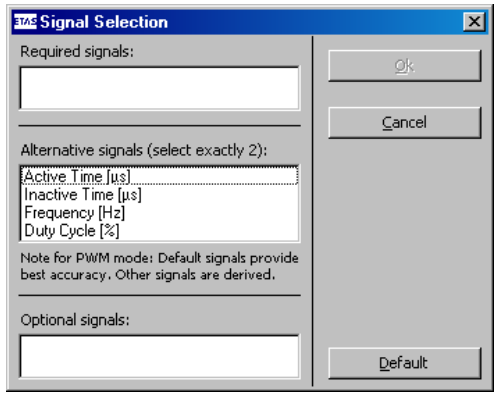
- タスク選択リストから、データ転送用に以下のタスクを選択します。

Group	Task
`Ch01Gen, Ch01Gen`	`T_1ms`
`Ch04Gen`	`T_10ms`

- **OK** をクリックします。
選択したタスクが“Task”列に表示されます。

3. シグナルを選択します。

   - 各行の“Signals”列をダブルクリックして“Signal Selection”ダイアログボックス を開きます。



   シグナルについての説明は、249 ページの「Signals」という項を参照してください。

   - 以下のシグナルを選択します。

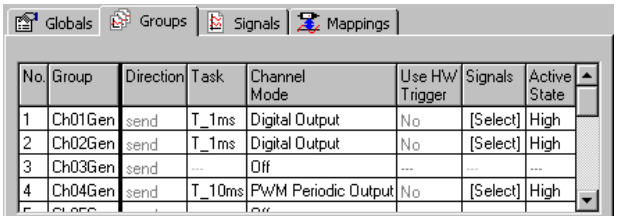
Group	Signals
`Ch01Gen`, `Ch02Gen`	デフォルト設定
`Ch04Gen`	`Frequency [Hz]`, `Duty Cycle [%]`

   - **OK** をクリックして選択を確定します。

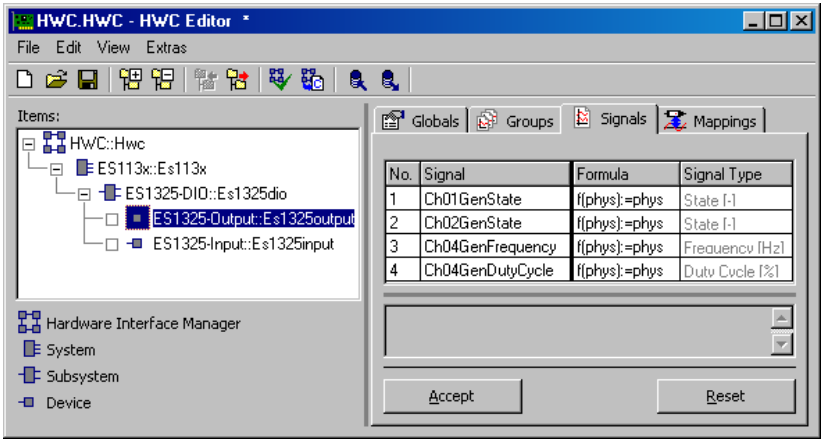
4. レベルを選択します。
   すべてのグループがデフォルト設定の `High` を使用するので、“Active State”列については何も変更する必要はありません。

   - すべての設定が終了したら、**Accept** ボタンで設定内容を保存します。

シグナルグループの設定が終わると、“Groups”タブの表示は以下のようになります。

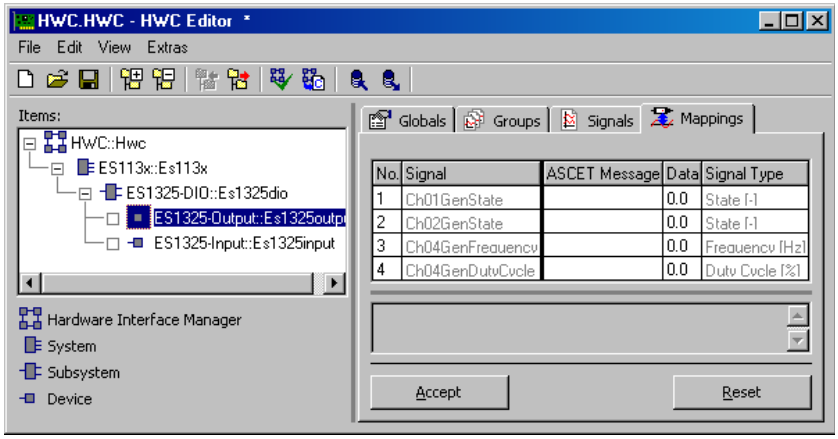


“Signals”タブ： ここには“Groups”タブで生成されたシグナルが表示されます。



このタブではシグナルの名前と式を編集できますが、サンプルプロジェクトにおいては、何も*変更する必要はありません*。

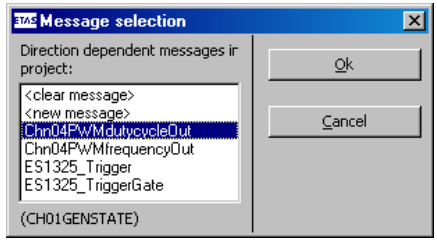
“Mappings” タブ： ここではプロジェクトのシグナルと ASCET メッセージを互いに関連付けます。“ASCET Message” 列のセルをクリックすると選択ダイアログが開きます。そのダイアログには、*Exported* 属性でしかもシグナルグループの転送方向と一致するメッセージ（ここでは *send メッセージ*）だけが表示されます。



シグナルに任意の ASCET メッセージを割り当てる：

- “Mappings” タブを選択します。
- 当該行の “ASCET Message” 列をダブルクリックします。

  “Message selection” ダイアログボックス が開きます。



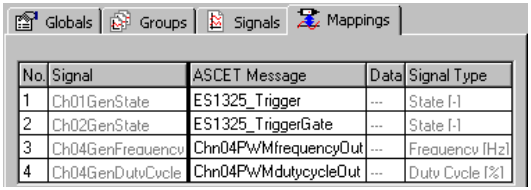
  ASCET プロジェクトに含まれるすべての Send メッセージが表示されます。

- 各シグナルについて、以下の ASCET メッセージを選択します。

Signal	ASCET Message
`Ch01GenState`	`ES1325_Trigger`
`Ch02GenState`	`ES1325_TriggerGate`
`Ch04GenFrequency`	`Chn04PWMfrequencyOut`
`Ch04GenDutyCycle`	`Chn04PWMdutycycleOut`

- **OK** をクリックします。
- HWC エディタの **Accept** ボタンをクリックして設定内容を保存します。

すべてのシグナルにメッセージか値を割り当てると、このタブの表示は以下のようになります。

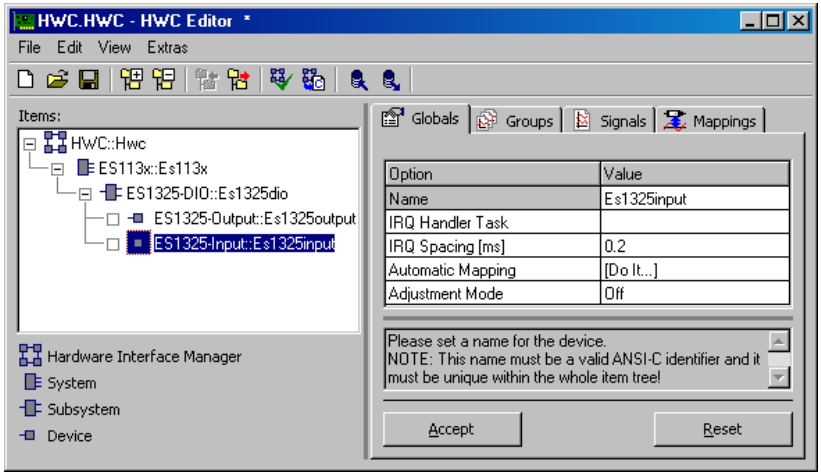


入力 ― ES1325-Input デバイス

“Globals” タブ： “Globals” タブでは、デバイス全体に共通するオプション設定を行います。

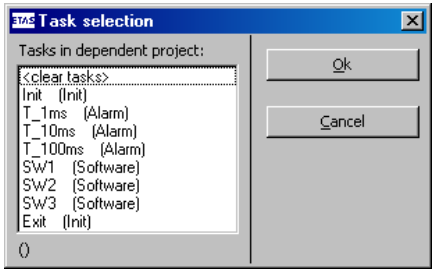
“Globals”タブの設定を行う：

- “Globals”タブを選択します。



- `IRQ Handler Task` オプションの隣にある空のボックスをダブルクリックします。

  “Task selection”ダイアログボックス が開きます。

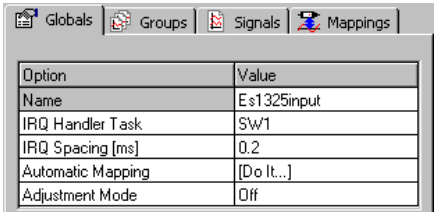


- `SW1` という Software タスクを選択します。

  他のオプションについては、デフォルト設定をそのまま使用します。

- **Accept** ボタンで設定内容を保存します。

このタブの表示は以下のようになります。

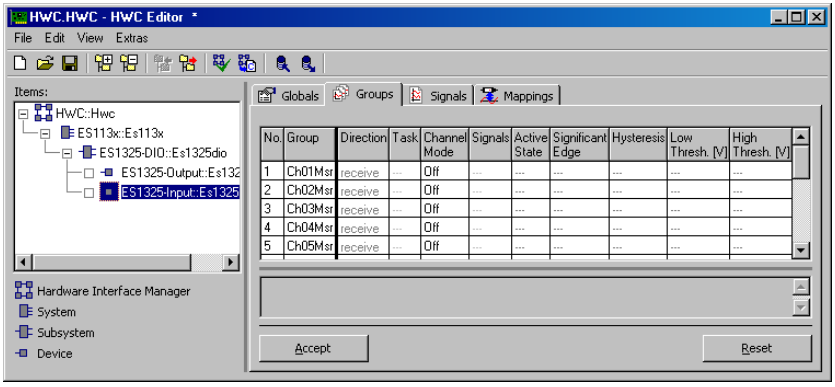


“Groups”タブ： “Groups”タブではシグナルグループごとの設定を行います。

`ES1325-Input` デバイスには、各入力シグナル（`Ch<n>Msr`、`<n>` = `01` ～ `16`）ごとにシグナルグループが割り当てられています。ここでは、使用される各グループについて、モード、タスク割り当て、生成されるシグナル、トリガタイプ（レベルまたはエッジ）を設定します。

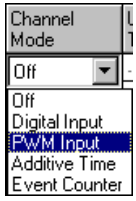
“Groups”タブの設定を行う：

- “Groups”タブを選択します。



シグナルグループ 4 について、以下の設定を行います。

1. チャンネルモードを選択します。

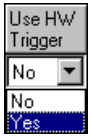


- 4 番目のシグナルグループの行の“Channel Mode”列をダブルクリックします。

  ドロップダウンリストが開きます。
- `PWM Input` モードを選択します。

  同じ行の他の列に、このモードのデフォルト値がセットされます。

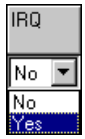
2. ハードウェアトリガの使用法について設定します。



- 同じ行の“Use HW Trigger”列をダブルクリックします。

  ドロップダウンリストが開きます。
- Yes を選択します。
- 同じ行の“IRQ”列をダブルクリックします。

  ドロップダウンリストが開きます。

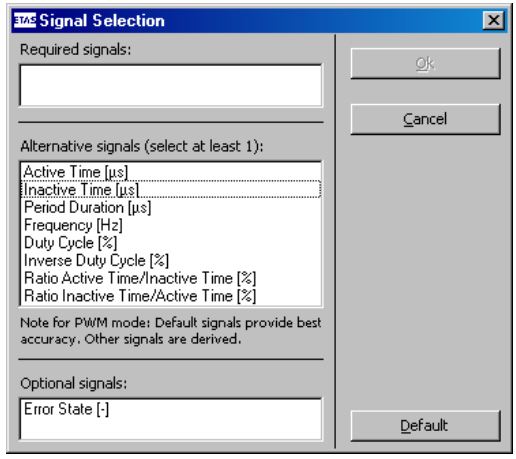


- Yes を選択します。

  “Task”列の内容がリセットされ、ロックされます。

3. シグナルを選択します。

- 同じ行の“Signals”列をダブルクリックして“Signal Selection”ダイアログボックス を開きます。

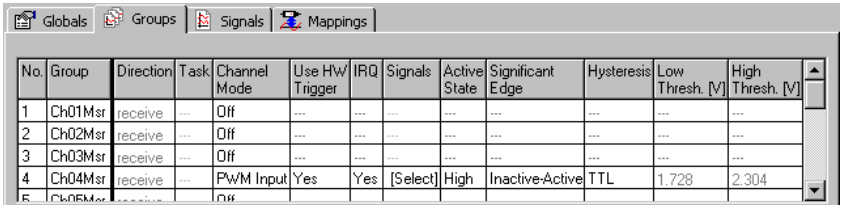


このダイアログボックス で、当該グループ用に生成されるシグナルを指定します。シグナルについての説明は、242 ページの「Signals」という項を参照してください。

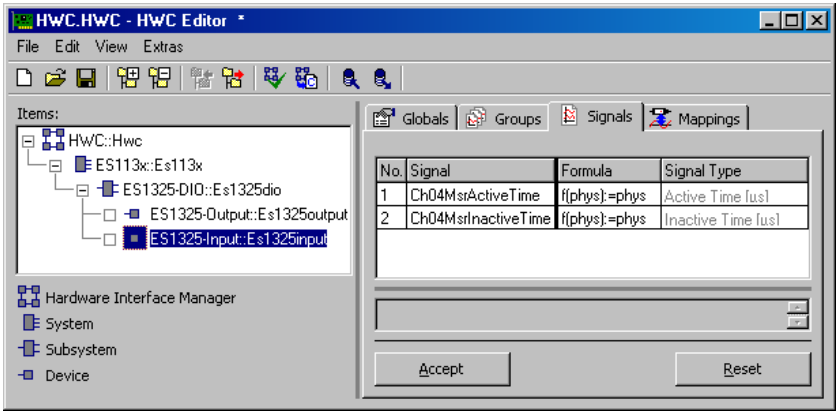
- Active Time [µs] と Inactive Time [µs] というシグナルを選択します。

  “Active State”列と“Significant Edge”列についてはデフォルト値を使用するので、これ以上設定する必要はありません。
- 必要な設定をすべて終えたら、HWC エディタの **Accept** をクリックして設定を保存します。

シグナルグループの設定が終わると、“Groups”タブの内容は以下のようになります。

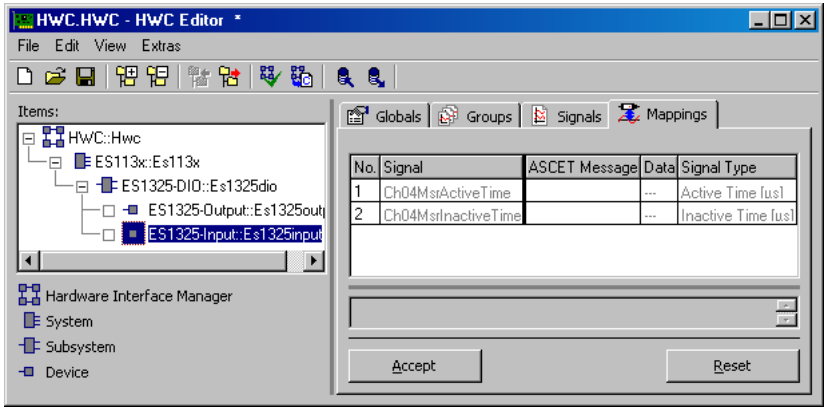


“Signals”タブ： ここには“Groups”タブで生成されたシグナルが表示されます。



このタブではシグナルの名前と式を編集できますが、サンプルプロジェクトにおいては、何も*変更する必要はありません*。

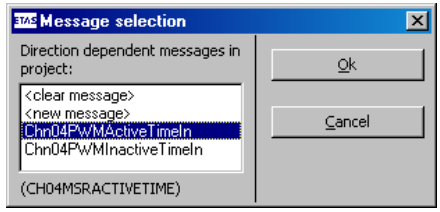
“Mappings” タブ： ここではプロジェクトのシグナルと ASCET メッセージを互いに関連付けます。“ASCET Message” 列のセルをクリックすると選択ダイアログが開きます。そのダイアログには、*Exported* 属性でしかもシグナルグループの転送方向と一致するメッセージ（ここでは *receive メッセージ*）だけが表示されます。



シグナルに任意の ASCET メッセージを割り当てる：

- “Mappings” タブを選択します。
- 各行の “ASCET Message” 列をダブルクリックします。

  “Message selection” ダイアログボックス が開きます。



  ASCET プロジェクトに含まれるすべての Receive メッセージが表示されます。

- 各シグナルについて、以下の ASCET メッセージを選択します。

Signal	ASCET Message
Ch04MsrActiveTime	Chn04PWMActiveTimeIn
Ch04MsrInactiveTime	Chn04PWMInactiveTimeIn

- **OK** をクリックします。
- HWC エディタの **Accept** ボタンをクリックして設定内容を保存します。

すべてのシグナルにメッセージが割り当てられると、このタブの表示は以下のようになります。

11.5.5 ハードウェアコンフィギュレーションの保存

作成したハードウェアコンフィギュレーションは、プロジェクトのファイルコンテナ内に保存したり、または DOS ファイル（`*.HWX`）として保存することができます。293 ページの 11.2.5 項を参照してください。

注記

サンプルプロジェクトの参考用コンフィギュレーションは、`ES1325.hwk` という名前で保存されています。このファイルを上書きしないように注意してください。

11.5.6 HWC モジュール用のコードの生成

続いて、HWC モジュールのコード生成を行います。コード生成の方法は、294 ページの 11.2.6 項を参照してください。

11.5.7 サンプルプロジェクトの実験

前項までで RTIO に関する設定がすべて完了したので、HWC エディタを閉じ、次にプロジェクトエディタで実験ターゲット用の標準的なコードを生成します。

注記

HWC モジュールをプロジェクトエディタのグラフィック表示に含めることはお勧めできません。HWC モジュールは RTIO のコード生成が行われるたびに変わるため、予期しない形で表示されてしまうためです。

実験ターゲット用のコードを生成する：

- プロジェクトエディタから **Component → Build** を選択して、プロジェクト全体のコードを生成します。
- **Component → View Generated Code** を選択して生成されたコードを表示し、内容を確認します。

オンライン実験を行う：

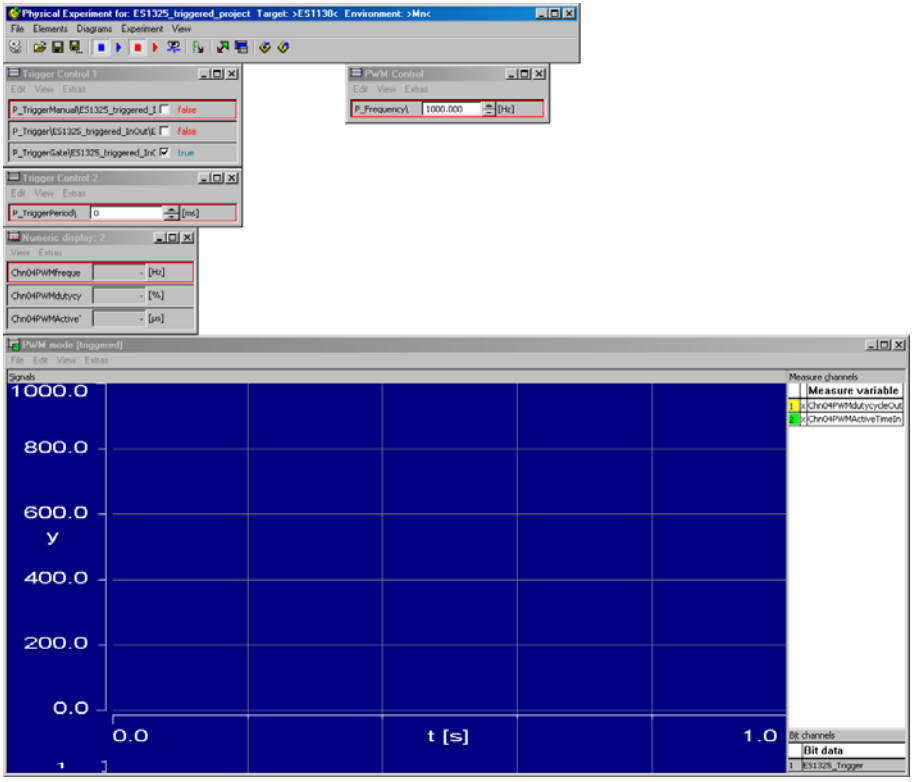
- ES1325 の入力ポートと出力ポートがまだ接続されていない場合は 317 ページの「接続」の項の記述に従って接続し、ES1000.x の電源をオンにします。

- プロジェクトエディタの“Experiment Target”コンボボックスから `Online (RP)` を選択します。

  `Offline (RP)` は、ターゲット上でオフライン実験を行うための設定です。

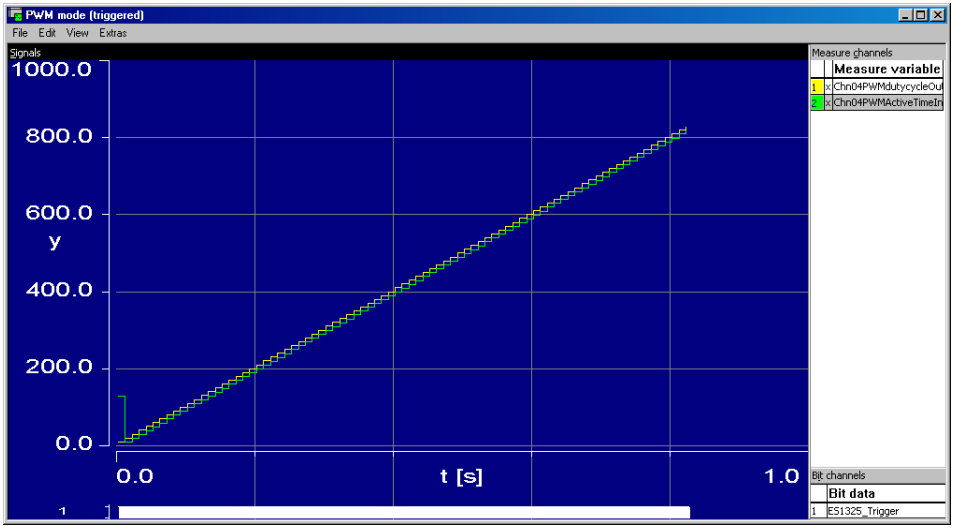
- **Component → Open Experiment** を選択します。

  “Physical Experiment”ウィンドウガ開き、あらかじめ定義されている実験環境（1 つのオシロスコープ、1 つの数値ディスプレイ、3 つの適合エディタで構成されます）が開きます。



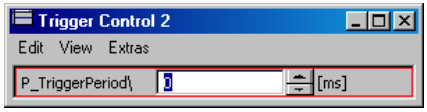
デフォルト設定では、第 1 のトリガシグナルと第 2 の固定されたトリガシグナルの値は、自動的に計算されるようになっています。つまりパラメータ `P_TriggerManual` は `false`、`P_TriggerGate` は `true` に設定されています。

“PWM mode (triggered)”というタイトルのオシロスコープには、`Chn04PWMdutycycleOut`（黄色いライン）という出力メッセージと`Chn04PWMActiveTimeIn`（緑のライン）という入力メッセージが表示され、さらに、`ES1325_Trigger`メッセージ（第 1 のトリガシグナルが格納されています）が下部のビットフィールドに表示されます。

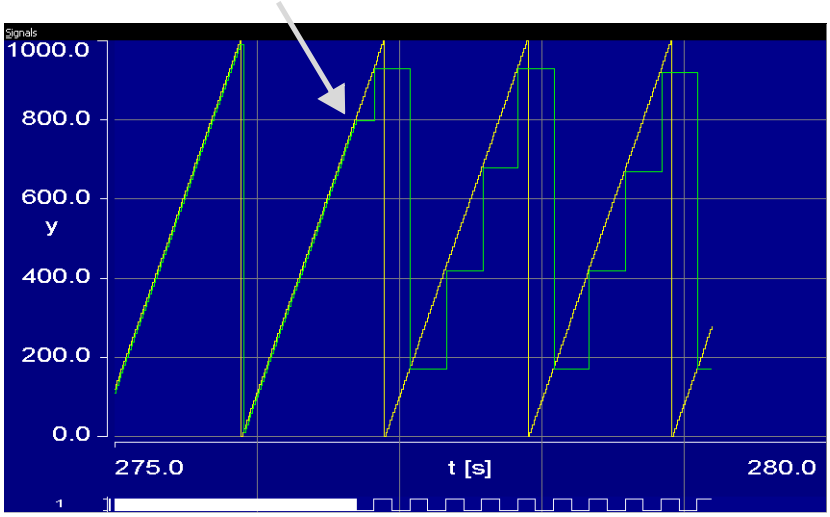


デフォルト値を用いて測定を開始すると、2 本のラインが、1 ステップ分タイミングがずれた状態で表示されます。

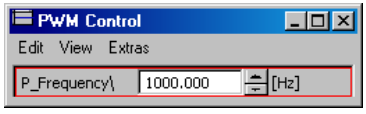
“Trigger Control 2”というタイトルの適合エディタに表示されている`P_TriggerPeriod`というパラメータの値を大きくすると、立ち上がりエッジ間の時間が長くなり、それに伴い`Chn04PWMActiveTimeIn`が読み込まれる頻度が少なくなります。



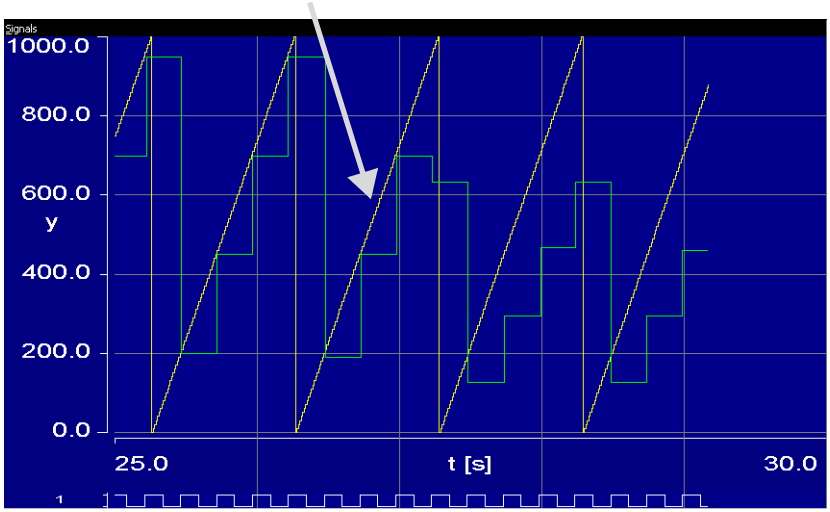
下図の例では、矢印のポイントにおいて P_TriggerPeriod パラメータが 0ms から 250ms に変更されたため、オシロスコープの下部に表示されるトリガシグナルの立ち上がりエッジの間隔が 250ms になり、それに伴って Chn04PWMdutycycleOutIn メッセージの読み取りの間隔も長くなり、ラインのステップが大きくなっています。



また、“PWM Control”適合エディタでパラメータ P_Frequency を変更すると、周波数が変わり、それに応じてアクティブ時間と非アクティブ時間も変わります。デューティーサイクルは変わりません。

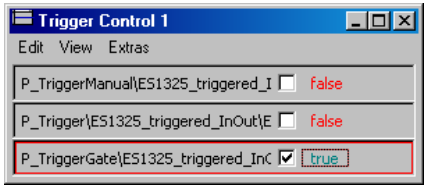


下図の例では、矢印のポイントにおいて周波数が 1000Hz から 1500Hz に変更されています。これによって `Chn04PWMdutycycleOut` のラインは影響を受けませんが、`Chn04PWMActiveTimeIn` のラインの最大値が小さくなりました。

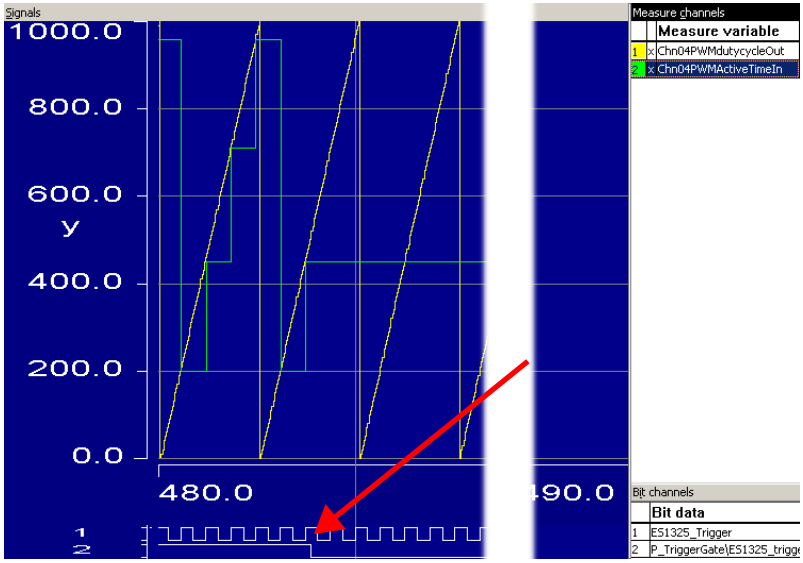


上図の場合、オシロスコープ上でショートカットメニューを開いて **Extras → Setup** を選択すると、“Display setup”ダイアログボックスが開き、その中の“Extent”フィールドには、x 軸の範囲が 5 秒に設定されています。

“Trigger Control 1”適合エディタでは、トリガをコントロールできます。



たとえば、P_TriggerGate パラメータの値を false にすると、第 2 のトリガシグナルが false になるので、Chn04PWMActiveTimeIn は読み取られなくなり、このラインの値は、最後に読み取られたときの値のまま保持されます。この場合も Chn04PWMdutycycleOut のラインは変わりません。

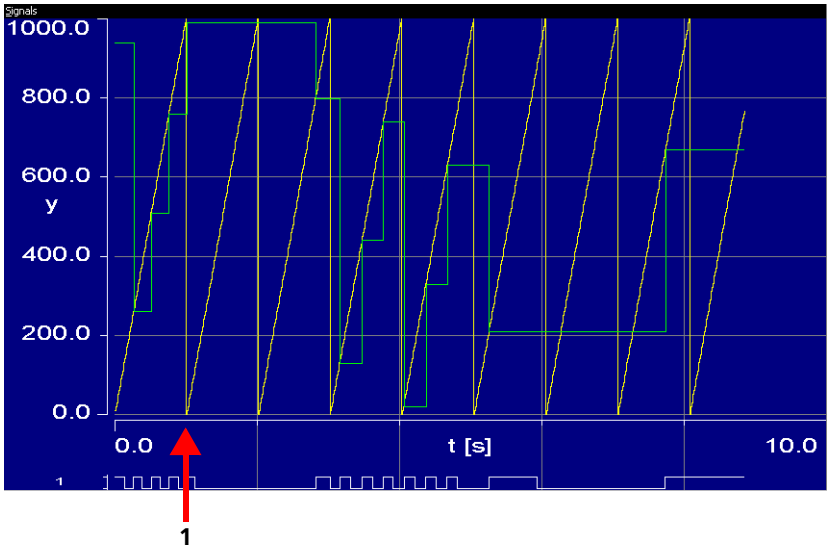


上図の場合、オシロスコープの x 軸の範囲は 10 秒に設定されていて、さらに変数 P_TriggerGate が追加されています。

P_TriggerGate の値を true に戻し、Chn04PWMActiveTimeIn の測定を続けてください。

また、同じ適合エディタ“Trigger Control 1”の P_TriggerManual パラメータの値を true にすると、第 1 トリガシグナルのマニュアルコントロールが有効になります。すると P_Trigger パラメータの値を false から true に変えるたびにトリガが発生し、同時に Chn04PWMActiveTimeIn メッセージが読み取られます。

下図の例では、ポイント 1 においてマニュアルトリガが有効になっています。`ES1325_Trigger` シグナルのラインは、`P_Trigger` がどのように変更されたかを示しています。`Chn04PWMActiveTimeIn` は、このトリガシグナルの立ち上がりエッジのたびに読み取られています。



上図の場合、オシロスコープの x 軸の範囲は 10 秒に設定されています。

12 ETAS ネットワークマネージャ

本章では、ETAS ネットワークマネージャを使用して ETAS ネットワークのコンフィギュレーション設定を行う方法を説明し、いつくかのネットワーク構成例もご紹介します。

12.1 概要

ASCET-RP からイーサネット経由でハードウェアシステムにアクセスするには、以下のいずれかの方法で行えます。

- 複数のネットワークアダプタ（イーサネットカード）を使用
  - 社内 LAN 用アダプタ
  - ETAS ハードウェア用アダプタ
- 1 つのネットワークアダプタを共有
  - 社内 LAN と ETAS ハードウェアとを自動的に切り替え可能

注記

ASCET-RP V5.5 以降、ETAS ハードウェアを PC に接続するための個別のネットワークアダプタは必要なくなりました。社内 LAN と ETAS ネットワークとで 1 つのネットワークアダプタを共有できます。

ETAS ネットワークマネージャで、ネットワークアダプタを ETAS ハードウェアに割り当てます。

ETAS ネットワークマネージャには、PC にインストールされているネットワークアダプタの一覧と、各アダプタの IP アドレス割り当て状況が表示されます。システム内に 2 つ以上のネットワークアダプタがインストールされている場合は、ここでどちらのアダプタを ETAS ハードウェア用に使用するかを選択できます。また、ETAS ハードウェア用に使用される IP アドレスの範囲を指定することもできます。

ネットワークアダプタを選択したり ETAS ハードウェア用のネットワーク環境を設定する際、管理者の権限は必要ありません。また ETAS ネットワークと社内 LAN との切り替えは、PC を再起動することなく行なえます。

注記

ネットワークマネージャでは、ネットワークアダプタのコンフィギュレーション自体を新しく作成したり変更することはできません。これらのことを行うには、管理者の権限が必要です（オペレーティングシステムのマニュアルを参考にしてください）。

12.2 ETAS ハードウェアのアドレッシング

ETAS ネットワークを使用すると、複数のデバイスを PC に接続することができ、同種のデバイスを複数接続することも可能です。ETAS ローカルネットワークに接続された各デバイスは、それぞれ異なる IP アドレスで認識されます。

ASCET-RP に組み込まれた IP マネージャがアドレスプールから 1 つのアドレスを引き出し、接続されたデバイスに割り当てます。

アドレスプールのアドレス範囲は、ETAS ネットワークマネージャを使用して設定します。

12.3 ネットワークアダプタのアドレス設定

12.3.1 ネットワークアダプタのアドレッシングタイプ

社内 LAN で使用されるネットワークアダプタのアドレッシングには、オペレーティングシステムの種類やネットワークアダプタのコンフィギュレーションによって、以下のようなタイプが使用されます。

オペレーティングシステム	ネットワークアダプタのアドレッシングタイプ			
	マニュアル設定	DHCP	DHCP+APIPA	DHCP + 代替 IP アドレス
Windows NT	○	○	—	—
Windows 98 SE	○	○	○	—
Windows 2000	○	○	○	—
Windows XP	○	○	○	○
	○：使用可、—：使用不可			

また ETAS ネットワークがサポートするネットワークアダプタのアドレッシングタイプは、以下のとおりです。

オペレーティングシステム	ネットワークアダプタのアドレッシングタイプ			
	マニュアル設定	DHCP	DHCP+APIPA	DHCP + 代替 IP アドレス
Windows NT	○	○	—	—
Windows 98 SE	○	—	○	—
Windows 2000	○	—	○	—
Windows XP	○	—	○	○
	○：使用可、—：使用不可			

1 つのネットワークアダプタを社内 LAN と ETAS ネットワークの両方で共有する場合、DHCP アドレッシングのみをサポートするネットワークアダプタは使用できません（ただし Windows NT の場合を除きます）。DHCP が使用できるのは、APIPA または代替 IP アドレスが併用されている場合のみです。

12.3.2 ネットワークアダプタアドレスのマニュアル設定

ネットワークアダプタのアドレスをマニュアル設定する方法は、オペレーティングシステムによって異なるため、ご使用のオペレーティングシステムのマニュアルを参考にしてください。

ネットワークアダプタのアドレスをマニュアル設定するには、管理者の権限が必要ですので、詳しくはシステム管理者の方にご相談ください。

ネットワークアダプタのアドレスをマニュアル設定して固定 IP アドレスを割り当てていた場合、社内ネットワーク に接続したまま ETAS ハードウェアの検索や初期化の操作を行ってしまうとことを防ぐため、IP アドレスが ETAS ハードウェアに割り当てられる前に警告が表示されるように、ネットワークマネージャで設定しておくことができます。

12.3.3 DHCP によるネットワークアダプタのアドレス設定

DHCP でのアドレッシングを行うには、DHCP サーバーが利用可能である必要があります。DHCP サーバーがなかったり利用不可能であった場合（ETAS ネットワークへの接続時など）、ネットワークアダプタの設定は行えません。

このため、各オペレーティングシステム（Windows NT を除く）には、ネットワークアダプタに代替 IP アドレスを自動的に割り当てるための以下のようなメカニズムが用意されています。

Windows 98 SE

Windows 98 SE では、社内 LAN には自動的に DHCP アドレスが使用され、ETAS ネットワークには APIPA アドレスが使用されます。PC 起動時に社内 LAN（DHCP ネットワーク）に接続されていた場合、ETAS ソフトウェア（ASCET-RP など）が起動されると、オペレーティングシステムによって、ネットワークアダプタは DHCP アドレスから APIPA アドレスに自動的に再設定されます。この切り替え処理が完了するまでには 60 秒ほどかかります。ETAS ネットワークを使用した後に PC を DHCP ネットワークに再接続すると、ネットワークアダプタは DHCP アドレスに再設定されます。この処理には 5 分ほどかかる場合がありますが、この時間は、同じ処理をマニュアル実行することによって短くすることができます。そのためには、MS-DOS コマンドウィンドウから **`ipconfig /renew`** コマンドを実行してください。

APIPA アドレスを併用しない DHCP でのネットワークアダプタのアドレス設定はサポートされていません。

Windows 2000

Windows 2000 では、DHCP サーバーへの接続が確立されているかどうか自動的に判定され、接続されていない場合は APIPA によって IP アドレスが自動的に割り当てられます。ETAS ネットワーク内では、常に APIPA アドレスが使用されます。社内 LAN（DHCP ネットワーク）から ETAS ネットワークへの切り替えを行う際は、オペレーティングシステムがネットワーク接続の切断を検知してネットワークアダプタの再設定を開始できるよう、10 秒間ほど接続を切断したままにしてください。オペレーティングシステムによる DHCP アドレスから APIPA アドレスへの再設定処理には、60 秒ほどを要します。その後、PC を再び DHCP ネットワークに接続する際には、DHCP サーバーへの接続が検出された時点で、すぐにネットワークアダプタに対して DHCP アドレスが再設定されます。

APIPA アドレスを併用しない DHCP でのネットワークアダプタのアドレス設定はサポートされていません。

Windows XP

Windows XP では、DHCP サーバーへの接続が確立されているかどうか自動的に判定され、接続されていない場合は、APIPA によって IP アドレスが自動的に割り当てられるか、またはユーザー定義の代替 IP アドレスが使用されます。ETAS ネットワーク内では、APIPA アドレスと代替 IP アドレスのいずれか使用されます。社内 LAN（DHCP ネットワーク）から ETAS ネットワークへの切り替えを行う際は、オペレーティングシステムがネットワーク接続の切断を検知してネットワークアダプタコンフィギュレーションの再設定を開始できるよう、10 秒間ほど接続を切断したままにしてください。オペレーティングシステムによる DHCP アドレスから APIPA アドレスまたは代替 IP アドレスへの再設定処理には、60 秒ほどを要します。その後、PC を再び DHCP ネットワークに接続する際には、DHCP サーバーへの接続が検出された時点で、すぐにネットワークアダプタに DHCP アドレスが再設定されます。

代替 IP アドレスや APIPA を併用しない DHCP でのネットワークアダプタのアドレス設定はサポートされていません。

Windows NT

Windows NT では、DHCP アドレスはある一定の時間（「リース期間」と呼ばれます）の間のみ割り当てられ、この期限を過ぎるとネットワークアダプタは未設定状態となります。そこで、ネットワークマネージャでこのリース期間を延長することにより、社内 LAN から切り離されても、延長された期間はネットワークアダプタの IP アドレスを使用できるようなります。リース期間は ETAS ソフトウェア

（ASCET-RP など）が最初にハードウェアをアクセスした時からカウントされます。リース期間を延長する処理には約 20 秒かかり、ETAS ソフトウェアが終了すると、リース期間はもとの設定時間に戻ります。

注記

リース期間の変更は、DHCP ネットワークから切り離される時間が、現在設定されているリース期間よりも長い場合にのみ意味を持ちます。IP アドレスの衝突を防ぐため、ETAS ソフトウェアを終了して DHCP ネットワークに戻す際は、ETAS ソフトウェアの終了後 20 秒間は、DCHP ネットワークに接続しないでください。

12.4 ユーザーインターフェース

12.4.1 “ETAS ハードウェア用のネットワーク設定（ページ 1）”ダイアログボックス

現在使用可能なネットワークアダプタについての以下の情報が表示されます。

- 名前

  ネットワークアダプタの名前です。このダイアログでは変更できません。
- IP アドレス

  ネットワークアダプタの IP アドレスです。このダイアログでは変更できません。
- サブネットマスク

  サブネットマスクの設定です。このダイアログでは変更できません。
- DHCP

  ネットワークアダプタが DHCP 用に設定されているかどうかを表わします。

  — Enabled

  ネットワークアダプタが DHCP 用に設定されています。

— Disabled

ネットワークアダプタが固定 IP アドレス用に設定されています。

- 代替の IP コンフィギュレーション

  DHCP 設定のネットワークアダプタの代替 IP アドレスが表示されます。ここの表示内容は、オペレーティングシステムによって異なります。

  — APIPA

  Automatic **P**rivate **IP** **A**ddressing：ネットワーク接続用の IP コンフィギュレーションの自動化を行います。

  — ---

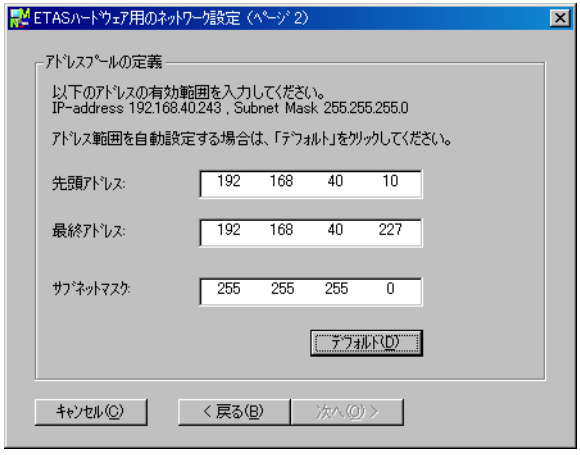
  代替 IP アドレスがありません。

  Windows NT の場合は、代替 IP アドレスはサポートされていません。

  — ユーザー定義

  代替 IP アドレスをユーザー定義することができます（Windows XP の場合）。

12.4.2 “ETAS ハードウェア用のネットワーク設定（ページ 2）”ダイアログボックス



各アドレスの値の変更は、キー入力によって直接編集するか、またはリストボックスからデフォルト設定を選択します。

以下のネットワークパラメータを設定できます。

- 先頭アドレス

  ETAS ハードウェア用の IP アドレス範囲の 1 番目のアドレスです。

- 最終アドレス

  ETAS ハードウェア用の IP アドレス範囲の最後のアドレスです。

- サブネットマスク
  使用するサブネットマスクです。

予約された IP アドレス

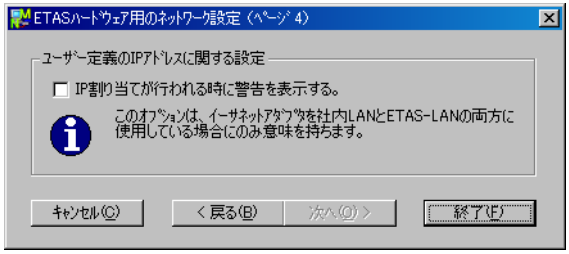
ETAS ハードウェアが現在使用している IP アドレス範囲（192.168.40.1 ～ 192.168.40.254、サブネットマスク 255.255.255.0）のうち、以下の IP アドレスは、ETAS の特定のハードウェア用に予約されています。

IP アドレス	ETAS ハードウェア
192.168.40.10	ES1120
192.168.40.11	ES1130
192.168.40.12	ES780
192.168.40.13	（予約済み）
192.168.40.14	LABCAR-RTPC
192.168.40.15	ES1135

これらのアドレスは特定のデバイスにのみ使用でき、他の ETAS ハードウェアには使用できません。アドレスプールを定義する際は、この点を考慮する必要があります。

12.4.3 “ETAS ハードウェア用のネットワーク設定（ページ 4）”ダイアログボックス

このダイアログボックスは、選択されたネットワークアダプタのアドレスがマニュアル設定されている場合にのみ開きます。



ここでは以下のオプションを設定します。

- IP 割り当てが行われる時に警告を表示する

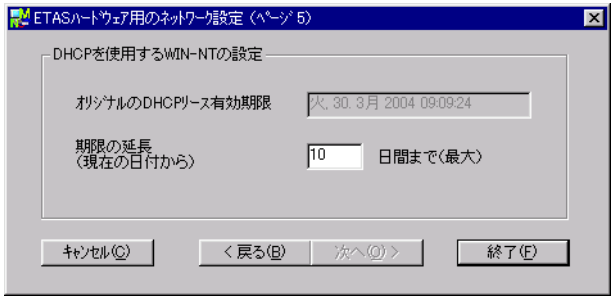
  このオプションがオンになっていると、ETAS ハードウェアデバイスに IP アドレスが割り当てられる際に、前もって警告メッセージが表示されます。

注記

この警告は、そのネットワークアダプタで、社内 LAN と、ETAS ネットワーク内の ETAS 測定モジュールの両方に接続する場合にのみ意味を持ちます。

12.4.4 “ETAS ハードウェア用のネットワーク設定（ページ 5）”ダイアログボックス

このダイアログボックスは、オペレーティングシステムが Windows NT で、かつ選択されたネットワークアダプタが DHCP 用に設定されている場合にのみ開きます。



ここでは以下のオプションを設定します。

- オリジナルの DHCP リース期限

  DHCP のリース期間が終了する時間を示します。このフィールドは変更できません。

- 期限の延長

  リース期間を延長する日数を指定します。ここには 365 までの整数を入力できます。

12.5 ETAS ハードウェア用のネットワークアドレスの設定

12.5.1 ネットワークアドレスの設定：固定 IP アドレスのアダプタの場合

ネットワークマネージャを起動する：

- Windows の**スタート**メニューから、**プログラム → ETAS → ASCET5.1 → ETAS Network Settings** を選択します。

  “ETAS ハードウェア用のネットワーク設定（ページ 1）”ダイアログボックスが開きます。

ネットワークアダプタを選択する：

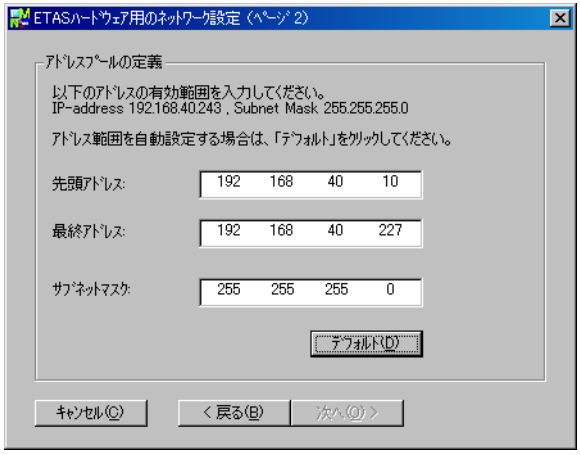
- **使用可能なネットワークアダプタ**フィールドで、社内 LAN と ETAS ネットワーク用に使用したいネットワークアダプタを選択します。

  ここでは、ETAS ネットワークがサポートするアドレッシングが指定されているネットワークアダプタしか選択できません。

  ここでネットワークマネージャを終了するには、**キャンセル** をクリックします。

- **次へ** をクリックして先に進みます。

  “ETAS ハードウェア用のネットワーク設定（ページ 2）”ダイアログボックスが開きます。



注記

DHCP 環境内でネットワークアダプタの設定を行っている場合は、上記のボタンの他に **終了** ボタンが表示されます。

ネットワークアドレスの設定：固定 IP アドレスを使用している場合

アドレスプールを設定する：

- “ETAS ハードウェア用のネットワーク設定（ページ 2）”ダイアログボックスで、“先頭アドレス”、“最終アドレス”、“サブネットマスク”のうち、変更したいフィールドを選択します。
- アドレスの値を直接キー入力します。

または

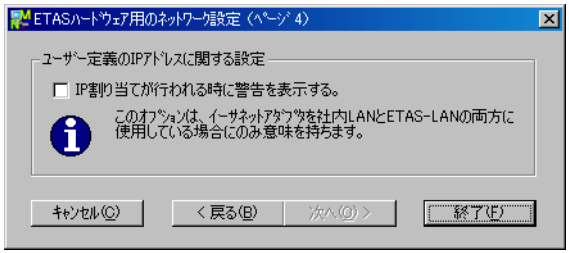
- **デフォルト** ボタンをクリックします。

  アドレス範囲とサブネットマスクの値は、ネットワークマネージャによって自動的に設定されます。その値を使用するか、またそれをさらに編集することもできます。

  設定内容を破棄してネットワークマネージャを終了するには **キャンセル** をクリックし、前のページに戻るには **戻る** をクリックします。

- **次へ** をクリックします。

  “ETAS ハードウェア用のネットワーク設定（ページ 4）”ダイアログボックスが開きます。



ユーザー定義の IP アドレスを設定する：

- ETASハードウェアへにIPアドレスが割り当てられる際に前もって警告メッセージが表示されるようにするには、**IP 割り当てが行われる時に警告を表示する** オプションをオンにします。

注記

この警告は、そのネットワークアダプタで、社内 LAN と、ETAS ネットワーク内の ETAS 測定モジュールの両方に接続する場合にのみ意味を持ちます。

  設定内容を破棄してネットワークマネージャを終了するには **キャンセル** をクリックし、前のページに戻るには**戻る** をクリックします。

- **終了** をクリックします。

  ネットワークアダプタの設定が終了し、ダイアログボックスが閉じます。変更された内容はすべて保存されます。

- 変更内容が有効になるように、ASCET-RP（またはその他の ETAS ソフトウェアアプリケーション）を再起動します。

  ASCET-RP でのハードウェア検索時または初期化時に自動的にネットワークマネージャが呼び出された場合は、再起動は必要ありません。

アドレスプールを設定する：

- “ETAS ハードウェア用のネットワーク設定（ページ 2)”ダイアログボックスで、“先頭アドレス”、“最終アドレス”、“サブネットマスク”のうち、変更したいフィールドを選択します。
- アドレスの値を直接キー入力します。

または

- **デフォルト** ボタンをクリックします。

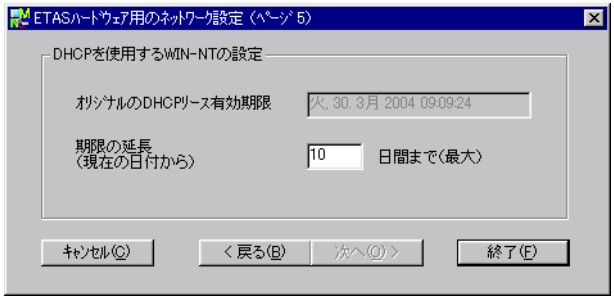
  アドレス範囲とサブネットマスクの値は、ネットワークマネージャによって自動的に設定されます。その値を使用するか、またそれをさらに編集することもできます。

1. DHCP 環境内において APIPA または代替 IP アドレスを使用しているネットワークアダプタの場合、以下のように操作して設定を終了します。
   - 終了 をクリックします。

     ネットワークアダプタの設定が終了し、ダイアログボックスが閉じます。変更された内容はすべて保存されます。
   - 変更内容が有効になるように、HSP およびその他の ETAS ソフトウェアアプリケーションを再起動します。

     HSP のハードウェア検索時または初期化時に自動的にネットワークマネージャが呼び出された場合は、再起動の必要はありません。
2. DHCP 環境内で使用されているネットワークアダプタを、Windows NT で使用している場合、以下のように操作して設定を続けます。
   - **次へ** をクリックします。

     “ETAS ハードウェア用のネットワーク設定（ページ 5)”ダイアログボックスが開きます。



Windows NT で DHCP を設定する：

- **期限の延長** フィールドにリース期間を延長する日数を入力します。

  設定内容を破棄してネットワークマネージャを終了するには **キャンセル** をクリックし、前のページに戻るには**戻る** をクリックします。
- **終了** をクリックします。

  ネットワークアダプタの設定が終了し、ダイアログボックスが閉じます。変更された内容はすべて保存されます。
- 変更内容が有効になるように、ASCET-RP（またはその他の ETAS ソフトウェアアプリケーション）を再起動します。

  ASCET-RP でのハードウェア検索時または初期化時に自動的にネットワークマネージャが呼び出された場合は、再起動は必要ありません。

12.6 イーサネット経由のハードウェアアクセスについてのトラブルシューティング

ここでは、イーサネットインターフェースで ETAS ハードウェアをアクセスする際に発生する可能性のある障害について説明します。

12.6.1 Windows 98 SE、2000、XP において APIPA が無効になる

IP アドレッシングの代替メカニズムである APIPA は、通常、Windows 98 SE、2000、および XP 環境においては有効になっていますが、時にはネットワークセキュリティポリシーによって無効となっている場合もあります。そのような場合、DHCP 設定のネットワークアダプタを ETAS ハードウェアのアクセスに使用することはできず、そのアダプタを選択すると ETAS ネットワークマネージャは警告メッセージを表示します。

無効になっている APIPA メカニズムを有効にするには、Windows のレジストリを編集する必要がありますが、これを行うには管理者の権限が必要です。ネットワーク管理者の方にご相談のうえ行ってください。

APIPA メカニズムを有効にする：

- Windows の **スタート** メニューから **ファイル名を指定して実行** を選択します。
- `regedit` と入力して **OK** をクリックします。

  レジストリエディタが開きます。

- 以下のフォルダを開きます。
  HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\
- **編集 → 検索** を選択して以下のキーを検索します。
  `IPAutoconfigurationEnabled`
- 見つかったキーの値をすべて `1` に変更し、APIPA メカニズムを有効にします。
  Windows のレジストリ内には、この名前のキーがいくつか含まれている場合があります。これは、TCIP/IP サービス全般、および個々のネットワークアダプタごとに APIPA メカニズムを無効にできるためです。
- レジストリエディタを閉じます。

APIPA メカニズムが無効になっていない場合、レジストリ内にこのキーは存在しません。

12.6.2 パーソナルファイアウォール

Windows XP には、パーソナルファイアウォールが組み込まれていますが、その他のシステムの場合でも、Symantec、McAfee、BlackIce、といったサードパーティのパーソナルファイアウォールが使われていることがよくあります。

このパーソナルファイアウォールは、ASCET-RP からのイーサネットハードウェアへのアクセスを妨害する場合があります。ハードウェアの自動検索を行った際、ハードウェアのコンフィギュレーションは正しく設定されているにもかかわらず、イーサネットハードウェアがまったく検出されない、というような場合、これは PC 上にファイアウォールソフトウェアがインストールされているためである可能性があります。

このような場合は、ASCET-RP での作業中にはファイアウォールソフトウェアを無効にするか、または IP アドレス 192.168.40.240 （ポート 18000 - 18005）に対してファイアウォールを開いておいてください。詳しくは、お使いのパーソナルファイアウォールの説明書を参照してください。

13 付録

本章では、ターゲットに依存する外部 C コードに関する注意事項（13.1「外部 C コード用コンパイラスイッチ」）と、実験ターゲット ES113x 用に使用できる ASCET-RP の API（**A**pplication **P**rogramming **I**nterface）関数を紹介します。これらの関数は、ASCET-RP と以下のコンポーネントとのインターフェースとして用いられます。

- ERCOS^{EK}（13.2 項）
- NVRAM（13.3 項）
- ウォッチドッグ（13.4 項）
- LED（13.5 項）
- その他（13.6 項）

13.1 外部 C コード用コンパイラスイッチ

ターゲット用の外部 C コードは、ターゲットに応じたスイッチで囲んでおくことが必要な場合があります。このため ASCET-RP V5.5 では以下のようなスイッチが用意されています。

- ES910
- ES1130
- ES1135
- ES113x（ES1130 および ES1135 用）

注記

`Prototyping` ターゲットが選択されている場合、`ES1135` スイッチによってテストコンパイルが行われます。

これらのスイッチは以下の構文で使用します。

```
#ifdef ES1135
...
/* ES1135 固有のコード */
...
#endif
```

13.2 API 関数（ERCOS^{EK}）

本項では、ASCET-RP の API（**A**pplication **P**rogramming **I**nterface）関数の仕様を詳しく説明します。これらのサービスルーチンは、アプリケーションと ERCOS^{EK} のインターフェースとして機能します。

各 API 関数の仕様は、以下の形式で記述されています。

_exampleRoutine	
機能	サービスの機能についての簡単な説明です。
構文	C 関数のプロトタイプ形式で構文が明記されています。
説明	ここには、サービスルーチンやその引数などについての詳細な説明、およびサービスルーチンを使用する際の注意事項が記載されています。
引数	引数として使用できる値の一覧です。（引数は、「説明」の行にまとめて記述されている場合もあります。）
戻り値	戻り値がある場合にはその型と値の範囲、および意味が明記されます。
使用例	関数の典型的な使用法を示しています。
参照	関連する関数のリストです。
ヒント	一部の関数については、さらに有用な情報を提供するヒントが付記されています。

次の表は、ASCET-RP において ES113x 実験ターゲット用のコマンドとして使用できる ERCOS^{EK} 用 API 関数の一覧です。各コマンドの詳細は、機能別に以降の項で説明します。

コマンド	機能	ページ
	アプリケーションモード	
DeclareAppMode	アプリケーションモードの外部宣言を行います。	136
SetNextAppMode	アクティブなすべてのタスクの処理を終了してから、指定されたアプリケーションモードに切り替えます。	140
	タスク	
DeclareTask	タスクの外部宣言を行います。	143
ActivateTask	SW タスクをアクティブ化します。	144
	システムタイム	
GetSystemTime	現在のシステムタイムを取得します。	151
GetSystemTimeLow	現在のシステムタイムの下位部分を取得します。	151
GetSystemTimeHigh	現在のシステムタイムの上位部分を取得します。	152
	割込み処理	

コマンド	機能	ページ
EnableAllInterrupts	すべての割込みをイネーブルにします。	179
DisableAllInterrupts	すべての割込みをディセーブルにします。	179

dT の問い合わせ		
GetDeltaT	dT の値を返します。	182

13.2.1 アプリケーションモード

アプリケーションモードのコンセプトを利用すれば、アプリケーションソフトウェアのさまざまな処理状態を効率的に管理することができます。各アプリケーションモードは、そのモードにおいてアクティブになる一連のタスクと、1 つまたは複数のタイムテーブル（オプション）により定義されます。たとえば、エンジン ECU 用アプリケーションの場合、アプリケーションモードには、通常運転（物理プロセス制御）、自動診断、フラッシュ EPROM プログラミングといったものが考えられます。アプリケーションモードは一度に 1 つだけアクティブにすることができます。

1 つのアプリケーションモードは 2 つのフェーズで構成されます。1 つめは初期化フェーズで、そこでアプリケーションの初期化ルーチンの処理が行われます。このフェーズでは割込みはディセーブルになっています。初期化が終わると割込みがイネーブルになり、実行フェーズが始動します。そこでは、アクティブ化されたタスクが、それぞれの優先度に応じて実行されます（＝スケジューリングされます）。

DeclareAppMode

機能　アプリケーションモードの外部宣言を行います。

構文
```
#define DeclareAppMode(Mode)
extern AppModeType Mode
```

説明　1 つのモジュール内でアプリケーションモードの切り替えが行われ、そのアプリケーションモードのディスクリプタが別のモジュール内で定義されている場合、そのディスクリプタの使用法が **DeclareAppMode()** で公開されている必要があります。
このサービスの機能と使用法は、変数の外部宣言と似ています。

使用例
```
extern uint excCtr;
extern uint randx;
DeclareAppMode(idleMode);
```

参照　DeclareTask

SetNextAppMode

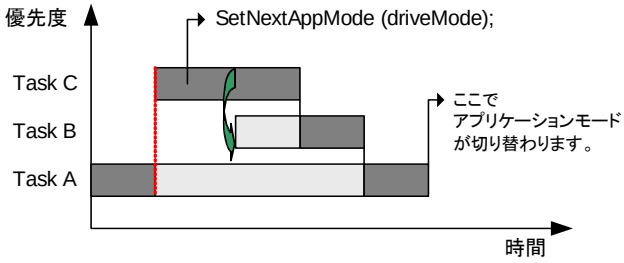
機能　アクティブなすべてのタスクの処理を終えてから、指定されたアプリケーションモードに切り替えます。

構文　`StatusType SetNextAppMode(AppModeType appMode)`

説明　**`SetNextAppMode()`** は、ポインタ appMode で参照されるアプリケーションモードへの切り替えをリクエストします。実行中のタスクがなくなると、つまりオペレーティングシステムがアイドル状態になると、オペレーティングシステムは直ちにこの切り替えを実行します。**`ChainTask()`** や **`RestartTask()`**（これらの API 関数は ASCET-RP ではサポートされていません）によるタスクアクティブ化は無視されます。ハードウェアタスクがスタートアップ時（初期化フェーズ）に初期化されるようになっている場合、それらのタスクは次のアプリケーションモード用に初期化されます。

戻り値　E_OK　リクエストは正常に処理されました。

使用例　**`SetNextAppMode`**`(driveMode);`



参照　—

13.2.2 タスク

ERCOS^{EK} には、2 種類のタスクがあります。1 つは **`ActivateTask()`** によりアクティブ化されるソフトウェアタスク（SW タスク）で、その実行は ERCOS^{EK} スケジューラにより管理されます。もう 1 つは割込みによりアクティブ化されるハードウェアタスク（HW タスク）で、その実行はプロセッサの割込み制御ロジック、つまりハードウェアにより行われます。

DeclareTask

機能　タスクの外部宣言を行います。

構文
```
#define DeclareTask(TaskID)
extern TaskType TaskID
```

DeclareTask

説明	あるモジュールで使用するタスクが別のモジュールで定義されている場合、その使用法が **DeclareTask()** で宣言されている必要があります。このサービスの機能と使用法は、変数の外部宣言と似ています。

使用例	`extern uint excCtr;`
	`extern uint randx;`
	`DeclareTask``(synchroSeq);`

参照	`DeclareAppMode`

ActivateTask

機能	SW タスクをアクティブ化します。	
構文	`StatusType ActivateTask(TaskType TaskID)`	
説明	**ActivateTask()** は、**TaskID** で指定された SW タスクの処理をオペレーティングシステムに要求します。タスクのアクティブ化に成功する（戻り値を参照してください）と、そのタスクはその優先度に応じて ERCOS^{EK} スケジューラによりスケジューリングされます。 BCC2 の仕様により 1 つのタスクを同時に複数個アクティブ化することが許されているため、タスクの現在のアクティブ化数が 1 より大きい場合、このタスクは一時的に FIFO バッファに格納されます。 **ActivateTask()** が正常に実行されなかった場合、ユーザー定義のエラーフック処理が実行されます。	
戻り値	`E_OK`	アクティブ化に成功しました。
	`E_OS_LIMIT`	アクティブ化は行われませんでした。（タスクのアクティブ化数が指定のリミット数に達していたか、あるいは指定された優先度レベルの FIFO バッファ内のタスク数が最大数に達していたため）
使用例	**`ActivateTask`**`(synchroSeq);`	
参照	—	

13.2.3 システムタイム

離散的なシステムタイムは、ERCOS^{EK} の時間管理の基礎となるものです。ハードウェアベースのシステムタイムを持たないターゲットの場合、システムタイムはオペレーティングシステムの起動と同時に 0 にセットされます。システムタイムは通常 2 ワード長でカウントされ、アラームサービスや ERCOS^{EK} タイムテーブルの基準時間として使用されます。システムタイムのオーバフローが発生するまでの時間は、CPU および使用されるハードウェアタイマの周波数によって異なります。アプリケーションモードが変化しても、システムタイムに割込みがかかったり、システムタイムがリセットされたりすることはありません。

システムタイムは、基礎となるタイマレジスタのカウント（tick）を単位としてカウントされます。**SYSTEM_TICK_DURATION** というマクロは、この 1 カウント（1 tick）の長さをナノ秒単位で返します。

GetSystemTime

機能	現在のシステムタイムを取得します。
構文	`TimeType GetSystemTime(void)`
説明	**GetSystemTime()** は、システムタイムを刻み単位で返します。システムタイムのワード長はシステムによって異なります（16 ビットシステムの場合は 32 ビット長、32 ビットシステムの場合は 64 ビット長です）。
戻り値	現在のシステムタイム
使用例	`TimeType now;` `now = GetSystemTime();`
参照	`GetSystemTimeLow, GetSystemTimeHigh`

GetSystemTimeLow

機能	現在のシステムタイムの下位部分を取得します。
構文	`TickType GetSystemTimeLow(void)`
説明	**GetSystemTimeLow()** は、現在のシステムタイム（刻みを単位とする値）の下位部分（32 ビット長のシステムタイムを使用するシステムの場合は下位 16 ビット、64 ビット長のシステムタイムを使用するシステムの場合は下位 32 ビット）を返します。
戻り値	現在のシステムタイムの下位部分
使用例	`TickType lowPartOfNow;` `lowPartOfNow = GetSystemTimeLow();`
参照	`GetSystemTime, GetSystemTimeHigh`

GetSystemTimeHigh

機能	現在のシステムタイムの上位部分を取得します。
構文	`TickType GetSystemTimeHigh(void)`
説明	**GetSystemTimeHigh()** は、現在のシステムタイム（刻みを単位とする値）の上位部分（32 ビット長のシステムタイムを使用するシステムの場合は上位 16 ビット、64 ビット長のシステムタイムを使用するシステムの場合は上位 32 ビット）を返します。
戻り値	現在のシステムタイムの上位部分
使用例	`TickType highPartOfNow;` `highPartOfNow = GetSystemTimeHigh();`
参照	`GetSystemTime, GetSystemTimeLow`

13.2.4 割込み処理

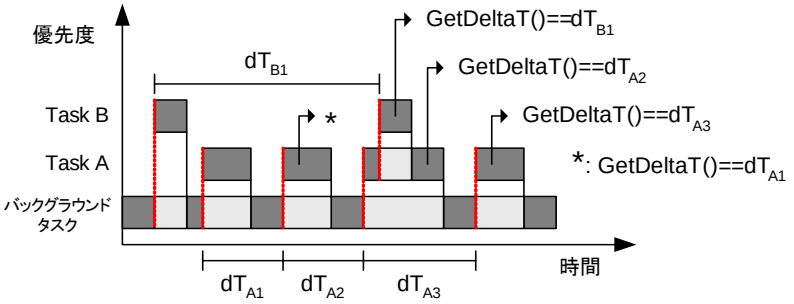
ERCOS^{EK} の割込み関連の API 関数のうち、ASCET-RP では割込みイネーブル／ディセーブルを制御するための関数がサポートされています。

EnableAllInterrupts	
機能	すべての割込みをイネーブルにします。
構文	`void EnableAllInterrupts(void)`
説明	**EnableAllInterrupts()** は、コントローラの割込みを、割込みマスクの操作なしにすべてイネーブルにします。 **DisableAllInterrupts()** が複数回呼ばれていた場合、それと同じ回数だけ **EnableAllInterrupts()** が呼び出されない限り割込みはイネーブルにならない、という安全な多重割込みディセーブル処理がサポートされています。
戻り値	なし
参照	`DisableAllInterrupts`

DisableAllInterrupts	
機能	すべての割込みをディセーブルにします。
構文	`void DisableAllInterrupts(void)`
説明	**DisableAllInterrupts()** は、すべての割り込みをディセーブルにします。
戻り値	なし
参照	`EnableAllInterrupts`

13.2.5 dT の問い合わせ

ERCOS^{EK} は、現在実行中のタスクの前回の処理開始から今回の処理開始までの時間を問い合わせるためのサービスルーチンを提供します（下図を参照してください）。返される時間は、このサービスを呼び出したタスクの dT の値です。



`GetDeltaT()` が返す dT は、以下のような積分計算を行う際に非常に有用です。

$$F(x)=\int_0^x f(T)\cdot dT$$

GetDeltaT	
機能	dT の値を返します。
構文	`TickType GetDeltaT(void)`
説明	**`GetDeltaT()`** は、タスクが 2 回連続して実行された間に経過した時間を返します。 **注記：**この時間がハードウェアタイマの幅の半分を超える値となった場合、返される値は信頼できません。 この関数は ERCOS^{EK} のデバッグモードでのみサポートされます。 ERCOS^{EK} アプリケーションのデバッグについての詳細は、『ERCOS^{EK} ユーザーズガイド』の「タスクモニタ内のデバッグ情報」という項を参照してください。
戻り値	dT の値（システムタイマのカウント数）
使用例	`TickType deltaT;` `deltaT = GetDeltaT();`
参照	—

13.3 API 関数（NVRAM）

NVRAM（不揮発性 RAM）に配置される NV 変数を扱う「NVRAM マネージャ」のデフォルトの機能は 6.2.1 項で説明されていますが、ASCET モデル（C コードコンポーネント）内から以下の API 関数を使用して、この挙動を変更することができます。

nvramInitModelVars

機能	NV 変数を初期化します。	
構文	`uint32 nvramInitModelVars(void)`	
説明	NVRAM の内容が有効であり、モデルとの整合性がある場合、NV 変数に NVRAM の内容を初期値として設定します。この初期設定は、NVRAM の定期的な更新を開始する前に、C コード、L1、または自動フラグを用いて、一度だけトリガできます。	
戻り値	`EC_NVRAM_SUCCESS`	成功
	`EC_NVRAM_NO_NV_VARIABLES`	モデル内に NV 変数がありません。
	`EC_NVRAM_INADMISSIBLE_USE`	関数が既にコールされています。
	`EC_NVRAM_INTERNAL_ERROR`	内部エラーが発生しました。
	`EC_NVRAM_NO_MATCH`	NVRAM の内容がカレントモデルとマッチしません。
使用例	–	
参照	`nvramCheckForInitializedVars`	

nvramSetUpdateInterval

機能	NVRAM の自動更新の間隔を設定します。	
構文	`uint32 nvramSetUpdateInterval(uint32 interval_sec)`	
説明	NVRAM を自動更新する間隔を設定します。システム負荷（現在設定されている整合性レベルに大きく依存します）が高くなると、実際の時間間隔はこの設定よりも長くなる可能性があり、10 回以上、要求された間隔よりも長い間隔で更新が行われると、実験環境内でワーニングが発行されます。	
引数：	`interval_sec`	更新間隔（単位は秒）。0 ～ 30 の値でなければなりません（0 を設定すると定期更新は行われません）。
戻り値	`EC_NVRAM_SUCCESS`	成功
	`EC_NVRAM_INVALID_ARG`	`Interval_sec > 30`
使用例	–	
参照	`nvramGetUpdateInterval`	

nvramGetUpdateInterval

機能　NVRAM の自動更新の間隔を取得します。

構文
```
uint32 nvramGetUpdateInterval(void)
```

説明　NVRAM の自動更新の間隔を取得します。

戻り値		
	`interval_sec`	更新間隔（単位は秒）。

使用例　–

参照　`nvramSetUpdateInterval`

nvramSetConsistencyLevel

機能　NV 変数のデータ整合性レベルを設定します。

構文
```
uint32 nvramSetConsistencyLevel
       (T_consistencyLevel level)
```

説明　NV 変数のデータ整合性レベルを設定します。
整合性なし：NVRAM の更新は、各変数の整合性を無視して行われます。
低レベルの整合性：NV 変数のうち、スカラ、ベクタ、マトリックス内のデータ整合性は保証されますが、特性カーブ／マップのデータの整合性は保証されません。
高レベルの整合性：NVRAM 内の NV 変数の更新は、アイドルタスクにおいて、すべての NV 変数の内容が中断なくローカルバッファに保存された場合のみ行われるので、すべての NV 変数の整合性が保証されます。

戻り値		
	`EC_NVRAM_SUCCESS`	成功
	`EC_NVRAM_INVALID_ARG`	引数 `level` は無効です。

引数	`level`	`NVRAM_NO_CONSISTENCY`
		`NVRAM_LOW_CONSISTENCY`
		`NVRAM_HIGH_CONSISTENCY`

使用例　–

参照　`nvramGetConsistencyLevel`

nvramGetConsistencyLevel

機能　NV 変数のデータ整合性レベルを取得します。

構文
```
T_consistencyLevel
       nvramGetConsistencyLevel(void)
```

説明　NV 変数のデータ整合性レベルを取得します。

戻り値		
	`NVRAM_NO_CONSISTENCY`	整合性なし
	`NVRAM_LOW_CONSISTENCY`	低レベルの整合性
	`NVRAM_HIGH_CONSISTENCY`	高レベルの整合性

使用例　–

参照　`nvramSetConsistencyLevel`

nvramEnableAutoUpdate

機能　NVRAM の内容の自動更新をオンにします。

構文　`uint32 nvramEnableAutoUpdate(void)`

説明　NVRAM の内容の自動更新をオンにします。これにより、Exit タスクにより行われる更新処理に加え、定期的な自動更新が行われます。

戻り値

`EC_NVRAM_SUCCESS`	成功

使用例　–

参照　`nvramDisableAutoUpdate, nvramCheckForAutoUpdate`

nvramDisableAutoUpdate

機能　NVRAM の内容の自動更新をオフにします。

構文　`uint32 nvramDisableAutoUpdate(void)`

説明　NVRAM の内容の自動更新をオフにします。これにより、定期的な更新と、Exit タスクにより実行される更新処理が行われなくなります。

戻り値

`EC_NVRAM_SUCCESS`	成功

使用例　–

参照　`nvramEnableAutoUpdate, nvramCheckForAutoUpdate`

nvramCheckForAutoUpdate

機能　自動更新がオンになっているかどうかを調べます。

構文　`uint8 nvramCheckForAutoUpdate(void)`

戻り値

`true`	自動更新はオンになっています。
`false`	自動更新はオンになっていません。

使用例　–

参照　`nvramEnableAutoUpdate, nvramDisableAutoUpdate`

nvramManualUpdateExit

機能　NVRAM の内容を、アプリケーションの最後に確実に更新します。

構文　`void nvramManualUpdateExit (void)`

nvramManualUpdateExit

説明　この関数を必ず Exit タスク内の最後のユーザープロセスの後に配置し、ユーザーアプリケーションモード終了時（Stop ERCOS ボタン押下時）に NVRAM の内容が最終的に確実に更新されるようにしてください。処理中にエラーが発生した場合には、実験環境内にエラーメッセージが表示されます。

使用例　–

参照　`nvramManualUpdateBackground`, `nvramManualUpdateBlocked`

nvramManualUpdateBackground

機能　NVRAM の内容の任意更新を開始します。

構文　`uint32 nvramManualUpdateBackground(void)`

説明　この関数は NVRAM の内容の任意更新を開始します。任意更新は定期的に行われる自動更新よりも優先されるので、この定期更新がすでに開始していた時に任意更新が要求されると、実行中の定期更新は中断されます。ただし、定期更新が途中で保留されている場合（つまり Idle タスクがプリエンプティブタスクからのこの関数呼び出しにより中断されていた場合）には、任意更新は開始できません。
実際の更新処理はバックグラウンドで（Idle タスク内で）行われるので、この関数は直ちに呼び出し元に戻ります。更新処理が完了したかどうかは、`nvramCheckRunningUpdate()` 関数により判断できます。
注記：自動更新がオンになっている間は、この関数は**使用しない**ことをお勧めします。

戻り値

`EC_NVRAM_SUCCESS`	成功
`EC_NVRAM_NO_NV_VARIABLES`	モデル内に NV 変数がありません。
`EC_NVRAM_FATAL_ERROR`	以前に重大なエラーが発生しています。
`EC_NVRAM_OVERFLOW`	NVRAM のオーバーフロー。NV 変数の数／サイズを少なくしてください。
`EC_NVRAM_UPDATE_RUNNING`	他の更新プロセス（任意更新または自動更新）が現在実行中です。任意更新の開始に失敗しました。

使用例　–

参照　`nvramManualUpdateBlocked`, `nvramManualUpdateExit`

nvramManualUpdateBlocked

機能　NVRAM の内容の任意更新を開始します（現在の優先度をブロックします）。

構文
```
uint32 nvramManualUpdateBlocked
       (uint32 timeoutUs)
```

説明　この関数は NVRAM の内容の任意更新を開始します。任意更新は定期的に行われる自動更新よりも優先されるので、この定期更新がすでに開始していた時に任意更新が要求されると、実行中の定期更新は中断されます。ただし、定期更新が途中で保留されている場合（つまり Idle タスクがプリエンプティブタスクからのこの関数呼び出しにより割り込まれていた場合）には、任意更新は開始できません。この関数は、すべての NV 変数の内容がローカルバッファに書き込まれるまで、またはタイムアウトが発生するまで、現在の優先度をブロックします。
実際の更新処理（ローカルバッファから NVRAM への書き込み）は、この関数から呼び出し元にリターンした後、Idle タスク内で続行されます（タイムアウトが発生しても行われます）。更新処理が完了したかどうかは `nvramCheckRunningUpdate()` 関数により判断できます。
更新処理中も割込みは保留されないので、更新処理より優先度の高いプリエンプティブタスクが更新処理に割込みをかける場合があり、このタスクが NV 変数の内容を変更すると、データの不整合が生じる可能性があります。
注記：自動更新がオンになっている際は、この関数は**使用しない**ことをお勧めします。

戻り値

`EC_NVRAM_SUCCESS`	成功
`EC_NVRAM_NO_NV_VARIABLES`	モデル内に NV 変数がありません。
`EC_NVRAM_FATAL_ERROR`	以前に重大なエラーが発生しています。
`EC_NVRAM_OVERFLOW`	NVRAM のオーバーフロー。NV 変数の数／サイズを少なくしてください。
`EC_NVRAM_UPDATE_RUNNING`	他の更新プロセス（任意更新または自動更新）が現在実行中です。任意更新の開始に失敗しました。

引数

`timeoutUs`	タイムアウト周期（µs）

使用例　–

参照　`nvramManualUpdateBlocked`, `nvramManualUpdateExit`, `nvramCheckRunningUpdate`

nvramCheckRunningUpdate

機能　NVRAM の任意更新が実行されているかどうかを調べます。

構文　`uint8 nvramCheckRunningUpdate(void)`

説明　この関数は、`nvramStartManualUpdateBackground` または `nvramStartManualUpdateBlocked` により開始された NVRAM 任意更新が、まだバックグラウンドで実行中であるかどうかを調べます。

戻り値

`false`	更新は実行されていません。（すでに終了したか、または開始に失敗しました。）
`true`	更新は今も実行中です。

使用例　–

参照　`nvramManualUpdateBackground`, `nvramManualUpdateBlocked`

nvramCheckForInitializedVars

機能　NV 変数が NVRAM の値で初期化されているかどうかを調べます。

構文　`uint8 nvramCheckForInitializedVars(void)`

説明　NVRAM に保存されていた値がモデル内の NV 変数の初期値として使用されたかどうかを調べます。初期化は、実験環境における自動更新により、あるいは C コードから呼び出された API 関数により、実行される可能性があります。

戻り値

`true`	NV 変数に NVRAM の値が初期設定されています。
`false`	NV 変数に、NVRAM の値ではなくモデルのデフォルト値が初期設定されています。

使用例　–

参照　`nvramInitModelVars`

nvramGetUpdateAgeMs

機能　前回の更新終了以降の経過時間を返します。

構文　`uint32 nvramGetUpdateAgeMs(void)`

戻り値　`updateAge`　時間（ミリ秒）

説明　この関数は、前回の更新（任意更新または自動更新）が終了してから経過した時間を返します。

使用例　–

参照　–

nvramClear

機能	NVRAM の内容を消去します。
構文	`uint32 nvramClear(void)`
戻り値	`EC_NVRAM_SUCCESS` 成功
説明	この関数は NVRAM の内容を消去します。メモリにはゼロが初期設定されます。
使用例	–
参照	–

13.4 API 関数（ウォッチドッグ）

ES1135 シミュレーションボードには、ハードウェアウォッチドッグ機能が搭載されています。この機能を使用するためのインターフェースは、ボードのファームウェアに含まれています。

ウォッチドッグ機能についての詳細は、6.2.2 項を参照してください。

13.4.1 ウォッチドッグの設定

wdSetSafetyMode

機能	ウォッチドッグをセーフティモードに切り替えます。
構文	`uint32 wdSetSafetyMode (uint32 event, uint32 period)`
説明	運用前モードまたは RSEF モードから、セーフティモードへの切り替えを行います。切り替え後は、電源をオフにしない限り、このモードを解除することはできません。 引数 **event** により、ウォッチドッグ満了時に実行するアクションが決まります。 `WD_EVENT_DISABLE` はウォッチドッグをオフにします。 `WD_EVENT_PPC750_RESET` は IBM 750GX シミュレーションプロセッサをリセットします。 `WD_EVENT_PPC750_INT` はシミュレーションプロセッサへの割込みをトリガします。 引数 **period**（ウォッチドッグが満了するまでの時間）には、0.25ms ～ 4096ms の値を設定できます。
戻り値	`EC_CFW_SUCCESS` 成功 `EC_CFW_WD_SAFETY_MODE` ウォッチドッグは既にセーフティモードになっています。 `EC_CFW_INVALID_ARG` `event` または `period` の値が無効です。

wdSetSafetyMode

引数	event	WD_EVENT_DISABLE WD_EVENT_PPC750_RESET WD_EVENT_PPC750_INT
	period	WD_PERIOD_4096MS WD_PERIOD_1024MS WD_PERIOD_256MS WD_PERIOD_64MS WD_PERIOD_16MS WD_PERIOD_4MS WD_PERIOD_1MS WD_PERIOD_0_25MS

使用例

```
uint32 period;
uint32 event;
uint32 retVal;
event = WD_EVENT_DISABLE;
period = WD_PERIOD_4096MS;
retVal = wdSetSafetyMode(event, period);
```

参照 wdSetPeriod, wdSetEvent

wdSetReducedSafetyMode

機能 ウォッチドッグを RSEF モードに設定します。

構文 `uint32 wdSetReducedSafetyMode(void)`

説明 運用前モードから RSEF（**R**educed **S**afety **E**nhanced **F**unction）モードへの切り替えを行います。
注記：この関数はブートローダ内でコールされます。従って、モデル実行開始時にはウォッチドッグはすでに RSEF モードになっているため、アプリケーションからこの関数を呼び出しても何も状態は変化しません。また、ブートローダはウォッチドッグイベントもディセーブルにします。ウォッチドッグの時間とイベントは、それぞれ wdSetPeriod と wdSetEvent により、任意に変更することができます。

戻り値		
	EC_CFW_SUCCESS	成功
	EC_CFW_WD_SAFETY_MODE	ウォッチドッグはセーフティモードになっています。これを解除することはできません。
	EC_CFW_WD_RSEF_MODE	ウォッチドッグはすでに RSEF モードになっています。

使用例 -

参照 wdSetPeriod, wdSetEvent

wdSetPeriod

機能　ウォッチドッグの監視時間を設定します。

構文　`uint32 wdSetPeriod(uint32 period)`

説明　ウォッチドッグの監視時間を設定します。この時間を過ぎてもウォッチドッグサービスがおこなわれないと、ウォッチドッグイベントが発生します。0.25ms ～ 4096ms の値を設定できます。

戻り値

`EC_CFW_SUCCESS`	成功
`EC_CFW_WD_SAFETY_MODE`	ウォッチドッグはセーフティモードになっています。`period` の修正はできません。
`EC_CFW_INVALID_ARG`	`period` の値が無効です。
`EC_CFW_WD_PRE_OP_MODE`	ウォッチドッグは運用前モードになっています。まず RSEF モードに切り替えてください。

引数

`period`	`WD_PERIOD_4096MS`
	`WD_PERIOD_1024MS`
	`WD_PERIOD_256MS`
	`WD_PERIOD_64MS`
	`WD_PERIOD_16MS`
	`WD_PERIOD_4MS`
	`WD_PERIOD_1MS`
	`WD_PERIOD_0_25MS`

使用例

```
uint32 period;
uint32 retVal;
period = WD_PERIOD_4096MS;
retVal = wdSetPeriod(period);
```

参照　`wdSetSafetyMode, wdSetEvent`

wdSetEvent

機能　ウォッチドッグタイマが満了したときに処理するイベントを設定します。

構文　`uint32 wdSetEvent(uint32 event)`

説明　ウォッチドッグが満了したときに実行されるアクションを選択します。
`WD_EVENT_DISABLE` は、ウォッチドッグをディセーブルにします。
`WD_EVENT_PPC750_RESET` は IBM 750GX シミュレーションプロセッサをリセットします。
`WD_EVENT_PPC750_INT` はシミュレーションプロセッサへの割込みをトリガします

戻り値

`EC_CFW_SUCCESS`	成功
`EC_CFW_WD_SAFETY_MODE`	ウォッチドッグはセーフティモードになっています。`event` の修正はできません。

wdSetEvent

	EC_CFW_INVALID_ARG	event の値が無効です。
	EC_CFW_WD_PRE_OP_MODE	ウォッチドッグは運用前モードになっています。まず RSEF モードに切り替えてください。
引数	event	WD_EVENT_DISABLE WD_EVENT_PPC750_RESET WD_EVENT_PPC750_INT

使用例

```
uint32 event;
uint32 retVal;
event = WD_EVENT_DISABLE;
retVal = wdSetEvent(event);
```

参照 wdSetSafetyMode, wdSetPeriod

13.4.2 ウォッチドッグサービス

wdService

機能 ウォッチドッグサービスを行います。

構文 Void wdService(void)

説明 ウォッチドッグサービスを行い、wdSetPeriod() によって設定された値をウォッチドッグタイマにセットします。

使用例 wdService();

参照 wdEnableAutoService, wdDisableAutoService

wdEnableAutoService

機能 自動サービスをオンにします。

構文 void wdEnableAutoService (void)

説明 自動ウォッチドッグサービス機能をオンにします。割込みがイネーブルになっている場合には、30ms 間隔でウォッチドッグサービスを行います。RTIO デバイスドライバにより、さらにサービスを行うこともできます。サービス実行はデフォルトではイネーブルになっています。

使用例 wdEnableAutoService();

参照 wdService, wdDisableAutoService

wdDisableAutoService

機能	自動サービスをオフにします。
構文	`void wdDisableAutoService(void)`
説明	自動ウォッチドッグサービス機能をオフにします。 **注記：**ウォッチドッグサービスをモデルで行う際には、適切な処理を行う必要があります。自動機能をディセーブルにすると RTIO 内部サービスコールもディセーブルになるので、注意が必要です。RTIO ドライバコール（特に Init と Exit）によって長時間のブロックが生じる可能性があるので、自動サービスをオンにする処理は、Init タスクと Exit タスクの中で行ってください。
使用例	`wdDisableAutoService();`
参照	`wdService, wdEnableAutoService`

13.4.3 割込み制御

wdIntEnable

機能	ウォッチドッグ割込み処理をイネーブルにします。
構文	`void wdIntEnable(void)`
説明	この関数はウォッチドッグ割込み処理をイネーブルにします。前もって `wdSetEvent()` を使用して、ウォッチドッグイベントを適切にマッピングしておいてください。`wdIntEnable()` は割込みの伝達だけに影響します。`wdIntPend()` は、ウォッチドッグ割込みがディセーブルになっていても使用できます。
使用例	`wdIntEnable();`
参照	`wdSetEvent, wdIntPend, wdIntDisable, wdIntAck`

wdIntDisable

機能	ウォッチドッグ割込み処理をディセーブルにします。
構文	`void wdIntDisable(void)`
説明	この関数はウォッチドッグ割込み処理をディセーブルにします。
使用例	`wdIntDisable();`
参照	`wdIntEnable`

wdIntPend

機能　割込みが処理待ち状態になっているかどうかを調べます。

構文
```
uint8 wdIntPend(void)
```

戻り値

false	処理待ち状態のウォッチドッグ割込みはありません。
true	ウォッチドッグ割込みが処理待ち状態になっています。

説明　ウォッチドッグ割込みが処理待ち状態になっているかどうかを調べます。前もって wdSetEvent() を使用して、ウォッチドッグイベントを適切にマッピングしておいてください。

使用例
```
if(wdIntPend() == true)
{
   intPollCount++;
   /* Reset Interrupt */
   wdIntAck();
}
```

参照　wdSetEvent, wdIntDisable, wdIntAck

wdIntAck

機能　ウォッチドッグ割込みに応答します。

構文
```
void wdIntAck(void)
```

説明　ウォッチドッグ割込みに応答するための関数です。ウォッチドッグタイマ（イベントをトリガした後は自動的に再起動されます）はこのコールの影響を受けません。ウォッチドッグタイマを初期化する必要がある場合には、先に wdService() を使用してください。

使用例
```
if(wdIntPend() == true)
{
   intPollCount++;
   /* Reset Interrupt */
   wdIntAck();
}
```

参照　wdSetEvent, wdIntDisable, wdIntPend

13.4.4 ウォッチドッグステータス

wdCheckReducedSafetyMode

機能　ウォッチドッグが RSEF モードになっているかどうかを調べます。

構文　`uint8 wdCheckReducedSafetyMode(void)`

説明　ウォッチドッグが Reduced Safety Enhanced Function（RSEF）モードで稼働しているかどうかを調べます。RSEF モードになっていれば、ウォッチドッグの設定をモデルから任意に修正できます。

戻り値
- `false`　ウォッチドッグはセーフティモードで稼働しています。
- `true`　ウォッチドッグは RSEF モードで稼働しています。

使用例
```
asdWriteUserDebug("Active = %u
       ReducedSafety = %u \n",
       wdCheckActive(),
       wdCheckReducedSafetyMode());
```
* `asdWriteUserDebug` については 13.6 項を参照してください。

参照　`wdSetSafetyMode, wdCheckActive`

wdCheckActive

機能　ウォッチドッグがオンになっているかどうかを調べます。

構文　`uint8 wdCheckActive(void)`

説明　ウォッチドッグが現在オンになっているかどうかを調べます。現在デバッガが ES1135 ボードに接続されている場合、ウォッチドッグはオフとなります。

戻り値
- `false`　ウォッチドッグは現在オンではありません。
- `true`　ウォッチドッグは現在オンになっています。

使用例
```
asdWriteUserDebug("Active = %u
       ReducedSafety = %u \n",
       wdCheckActive(),
       wdCheckReducedSafetyMode());
```
* `asdWriteUserDebug` については 13.6 項を参照してください。

参照　`wdSetSafetyMode, wdSetEvent, wdCheckReducedSafetyMode`

13.5 API 関数（ES1135 LED）

ES1135 シミュレーションコントローラには設定可能な LED が 3 つあります。これらについては、6.2.3 項に簡単に説明されています。この LED は、以下のインターフェースを使用して制御します。

userLed[n]On

機能　LED[n] をオンにします。

構文
```
void userLed1On(void)
void userLed2On(void)
void userLed3On(void)
```

説明　指定の LED をオンにします。

使用例
```
userLed1On();
```

参照　userLed[n]Off, userLed[n]Toggle

userLed[n]Off

機能　LED[n] をオフにします。

構文
```
void userLed1Off(void)
void userLed2Off(void)
void userLed3Off(void)
```

説明　指定の LED をオフにします。

使用例
```
userLed1Off();
```

参照
```
userLed[n]On
userLed[n]Toggle
```

userLed[n]Toggle

機能　LED[n] のオン／オフを切り替えます。

構文
```
void userLed1Toggle(void)
void userLed2Toggle(void)
void userLed3Toggle(void)
```

説明　指定の LED のオン／オフを切り替えます。

使用例
```
userLed1Toggle();
```

参照　userLed[n]Off, userLed[n]On

13.6 API 関数（その他）

さらにいくつかの API 関数を使用できます。

asdWriteUserError

機能	ASCET モニタウィンドウにコメントを出力します。
構文	ANSI-C 関数の `printf` と同じです。
説明	ASCET モニタウィンドウにユーザーメッセージを表示します。
使用例	`uint8 number = 1;` `asdWriteUserError("Example %u \n", number);`
参照	`asdWriteUserDebug`

asdWriteUserDebug

機能	ASCET “Target Debugger”ウィンドウにコメントを出力します。
構文	ANSI-C 関数の `printf` と同じです。
説明	ASCET “Target Debugger”ウィンドウにユーザーメッセージを表示します。
使用例	`uint8 number = 1;` `asdWriteUserDebug("Example %u \n", number);`
参照	`asdWriteUserError`

索引

G

H

I

L

N

し

た

て

ね

は

ひ

ふ

め

わ