

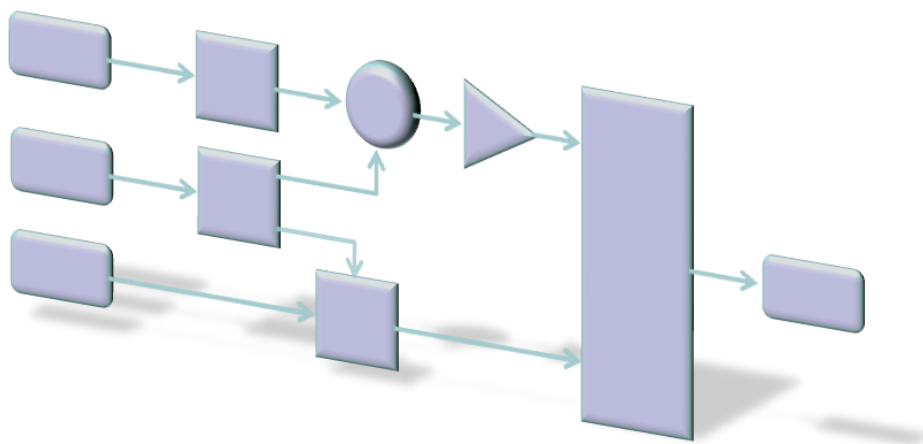
平成 23 年度文部科学省委託

東日本大震災からの復旧・復興を担う専門人材育成支援事業



MIBD
Model-Based Development

「モデルベース開発入門」



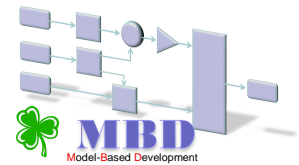
目次

1章	はじめに	5
1章1節	要求の多様化・高度化	5
1章2節	機能要求の変化	6
1章3節	開発期間の変化	7
	この章のまとめ	8
2章	組込開発の現状	9
2章1節	電子制御が脇役から主役に	9
2章2節	システムの複雑化と開発における課題	10
3章	MBD (Model - Based Development)	13
	MBDとは	13
	MBDを表すキーワード	15
4章	MBDの特徴	17
4章1節	機能のモデル化	17
4章2節	シミュレーション (実行可能な仕様書)	18
4章3節	自動コード生成	20
4章4節	フロントローディング	22
	・従来開発では試作で確認する	22
	・MBDではモデルで確認する	23
	・開発全体のフロントローディング	24
4章5節	コンカレント開発	25
	この章のまとめ	26
5章	JMAAB	27
5章1節	JMAABとは、	27
5章2節	MBD技術者の育成	28
5章3節	ETSS-JMAAB	29
6章	MBDツールとしてのMATLAB/Simulink	34
6章1節	MATLAB	34
6章2節	MBDツールに触れてみよう。	36
	6. 2. 1. モデルを描いてみよう (前準備)	36

• MATLAB を起動してみよう。	36
• Simulink を起動してみよう。	37
• モデルを描くキャンバスを用意しよう。	38
6. 2. 2. モデルを描いてみよう (ブロックの描画)	39
• 描画するブロックを選択しよう。(ブロックの選択)	39
• キャンバスにブロックを配置しよう。(ブロックの描画)	40
• キャンバス上に既にあるブロックをコピーしよう。	41
6. 2. 3. モデルを描いてみよう (ブロック間の結線)	42
• ブロック同士を結線しよう。(ブロック間の結線)	42
• 結線を削除するには?	43
• 結線を分岐させるには?	43
6章3節 シミュレーションしてみよう	44
6. 3. 1. モデルを動かしてみよう (シミュレーション)	44
• シミュレーションを実行します。	44
• パラメータを変更しよう。	45
• 変更結果を確認しよう。	46
6. 3. 2. ブロックを変更し、動作を確認しよう。	47
• ブロックを変更しよう。	47
6. 3. 3. $(4+3)/(2+1)$ の余りを求めるモデリングをしよう。(課題1)	50
6章4節 出力が変化するブロックを使ってみよう。	51
6. 4. 1. 出力が変化するブロックを使ってみよう。	51
• 出力を変化させてみよう。	51
6. 4. 2. 出力が変化するブロックを合成してみよう。	53
• 複数のブロックの出力を合成してみよう。	53
• 合成結果を確認してみよう。	55
• 変更後の出力を確認しよう。	56
6章5節 いろいろなブロックを使ってみよう	57
• カウンターモデルを作成してみよう。	57
• 正弦波を出力してみよう。	58
• 信号を多重化してみよう。	59
• 信号名を入力しよう。	60
• グラフ名を記述しよう。	61
• 「Scope」ブロックのパラメータを変更するには?	62
• 任意の波形を作成してみよう。	63
• コンフィギュレーションパラメータを設定しよう。	64
7章 実際の開発とモデリングとの関係	65

7章1節	機能と制御システムモデルの関係	65
7章2節	実際の制御システムの構成（自動車）	66
7章3節	制御システム（制御装置）における二種類の制御	67
7章4節	制御装置（コントローラ）	68
7章5節	フィードバック制御	69
8章	ON/OFF 制御モデルのモデリング	70
8章1節	ON/OFF 制御	70
8章2節	ON/OFF 制御モデルのモデリング	71
8.2.1.	簡易オートエアコンのサンプルモデルを開いてみよう。	71
8.2.3.	簡易オートエアコンの制御仕様	72
8.2.4.	プラントの解説	73
8.2.5.	簡易オートエアコンのシミュレーション（シナリオ）解説	74
8.2.6.	簡易オートエアコンのシミュレーション（観測）解説	75
8章3節	ON/OFF 制御を実際にモデリングしてみよう。（課題2）	76
8章4節	簡易オートエアコンの省エネ化（課題3）	77
9章	PID 制御モデルのモデリング	78
9章1節	PID 制御	78
9.1.1.	PID 制御の基本式	78
9.1.2.	PID 制御の基本式のモデリング	79
9章2節	PID 制御モデルのモデリング	80
9.2.1.	速度制御のサンプルモデルを開いてみよう。	80
9.2.2.	PID モデルの構成の説明	81
9.2.3.	PID モデルのモデリング	82
9.2.4.	コントローラとプラント	82
9章3節	PID 制御を実際にモデリングしてみよう。（課題4）	83
9.3.1.	PID 制御モデルのライブラリとパラメータ	83
9.3.2.	PID 制御モデルを検証してみよう。	84
付録	よく使われるブロック	85
あとがき		112

1章 はじめに



現在の自動車に求められる機能は、複雑化、高度化しています。しかし、それら機能を開発する開発力は、開発期間の短期化に伴い限界を迎えつつあります。

1章1節 要求の多様化・高度化

自動車メーカーは、消費者からの様々な要求に応え続けています。その結果、昔と比べて、自動車はとても高性能になり、新製品が発売される間隔も短くなりました。

自動車は、機能の塊です。自動車自体に求められている機能（基本機能）こそ、『走る・曲がる・止まる』の3種類ですが、これらを満たした上で、安全で快適なドライブを提供するために、実に様々な機能が自動車に搭載されています。

例えば、ABS、パワーウィンドウ、パワースライドドア、電動サンルーフ、etc.

これらの機能は、先の基本機能に満足した顧客が新たに産み出した要求です。現在も、新しい要求に応えようと新機能の開発が進んでいます。また、それらが実現した後も、新しい要求は生まれ、それを実現するための開発が進んでいきます。



図 1-1 : 自動車の機能

1章2節 機能要求の変化

自動車に求められる機能は、年々、より複雑なものに変化しています。

初期の要求は、走る、曲がる、止まる、の3点でした。これらは、自動車の基本機能として定義されています。新しい機能は、これまでよりも高度な要求を実現します。

例えば、センサーで周囲の状況を監視し、自律的に加減速やハンドル操作を行うことで運転者の負担を和らげます。また、周囲の状況を知ることによって事故を未然に回避し、より高い安全性を確保します。このような高度な要求に、高度な機能を開発することで応えています。

ドライバーの要求は、基本機能に満足していたころと比べると複雑かつ高度な機能を要求するようになりました。

重いハンドル操作を楽にするパワーステアリング機能や、衝突時の衝撃を和らげるエアバッグ機能などは、現在販売されている自動車のほとんどに備わっている程、当たり前の機能となっています。そうすると、衝突しそうになる前にブレーキがかかるなど、より高度な機能が新しい開発対象に選ばれます。



図 1-2 : 要求の変化

1章3節 開発期間の変化

消費者の要求は、時と共に次々と変わっていきます。その時に求められるものに素早く対応するため、自動車メーカーは、品質を維持した上で開発サイクルを短縮させています。

自動車メーカーは、互いに競争する立場にあります。安さや品質の競争の他、近年では開発サイクルを早め、新しい製品をより早く市場に出す傾向が強まっています。

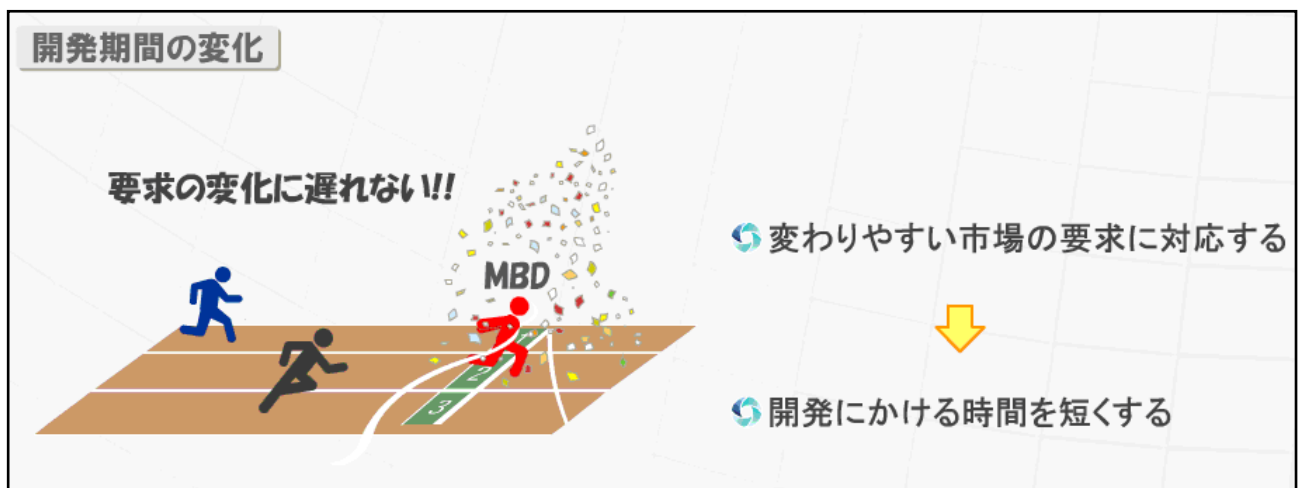


図 1-4：開発期間の変化

この章のまとめ

求められるものが変わった

- 高性能・多機能なものを作る

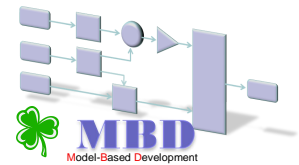
要求の変化に遅れない

- 開発期間を短くする



モデルベース開発（Model-Based Development : MBD）は、
複雑化した機能を明確にし、求められる納期に適した開発プロセスを実現します。

2章 組込開発の現状



2章 1節 電子制御が脇役から主役に

『日経 Automotive Technology』2008年7月号によると、

「車の競争力の源泉が、ハードウェアからソフトウェアに急速に変わりつつある。ユーザーは電子制御機能に付加価値を求め、開発現場は増大する負荷に対し、組織改正や海外への開発委託などで対応を急ぐ。」と記載しています。

また、「重要性を増す電子プラットフォーム」と題して、

「クルマの競争力がハードウェアからソフトウェアに比重を移しつつある流れは、クルマのプラットフォームの位置づけからも明らかだ。

これまでプラットフォームと言えば、パワートレインやサスペンション形式、フロア周りの構造など車両の機構的な基盤を指していた。しかし、電子装備が急速に増加している現在、電子システムの基盤となる電子プラットフォーム（PF）がクルマの性能を左右するようになってきた。」と記載しています。

出典：『日経 Automotive Technology 2008年7月号 p74』から転載

2章2節 システムの複雑化と開発における課題

電子制御システム開発力向上の課題

機能要求は、信頼性、コスト、耐久性、形状の要件を高次元でバランスさせて造ることが要求されます。かつ、システム構成は複雑で、多数のシステム間の連携、機能統合、適正分担が要求されます。

複雑な電装システム開発を、如何に信頼性を落とさずに短期間で実施できるかが最重要課題です。

ソースコード量の肥大化

高度な機能がひとつ追加されると、それに関わる部品全ての制御ソフトウェアに変更が加わります。

これは、搭載されるソフトウェアが増えることに繋がります。また、複数の装置が連携して機能を制御するため、通信用のソフトウェアを追加する必要があります。

結果として、機能を追加すればするほど、また、制御対象を連動させるほど、自動車に搭載されるソフトウェアの量も増えます。

ここで例に挙げている自動車は、ある企業が発売した高級ブランド自動車です。

この自動車には、一台当たり700万行のソースコードが搭載されています。カーナビゲーションシステムのソースコードを加えると、実に1000万行を超えるソースコードが搭載されています。

現在の高級車は、全てを1人のエンジニアが読むことができない程大規模なソフトウェアが搭載されています。



図 2-2 : ソースコード量の肥大化

従来の開発手法の行き詰まり

開発効率を上げて、開発スピードを早くする必要がありますが、この効率化にも限界があります。従来の開発手法にも改善できる点は残っています。

しかし、それにより得られる効果だけでは追いつけないほど、開発量の増加は著しくなっています。

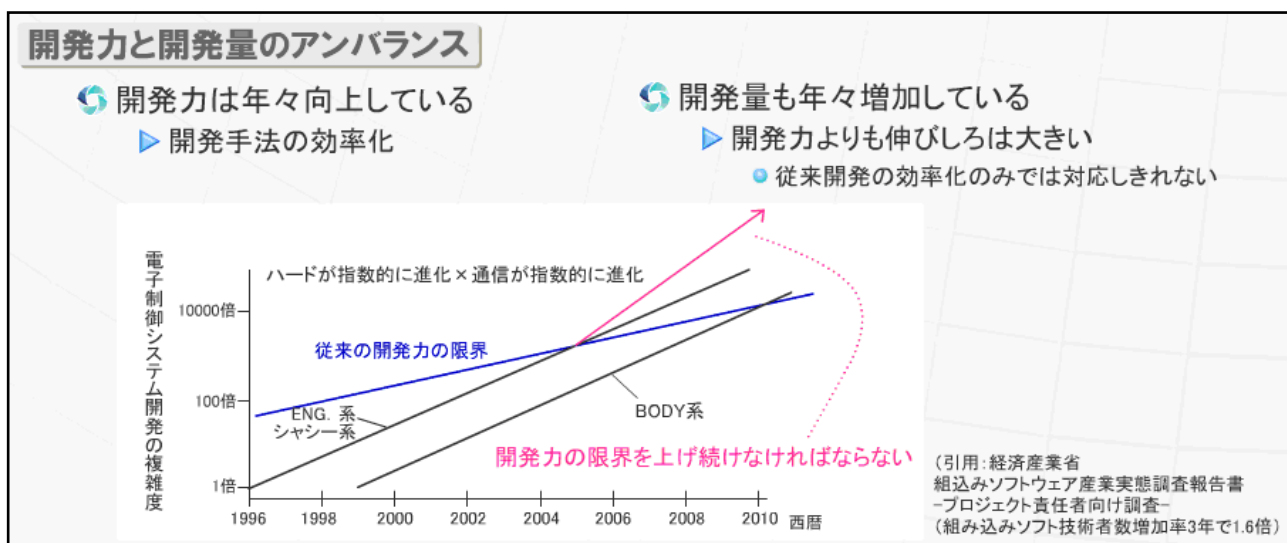
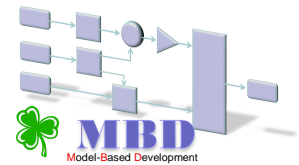


図 2-3 : 開発力と開発量の変化

開発力の限界を上げるには、従来開発を効率化する以外にも、何か手を打つ必要があります。

3章 MBD (Model - Based Development)



MBD とは

MBD を簡単に表現すると、” モデルを用いた開発 “であり、
” 協調開発 “を実現するための開発手法です。

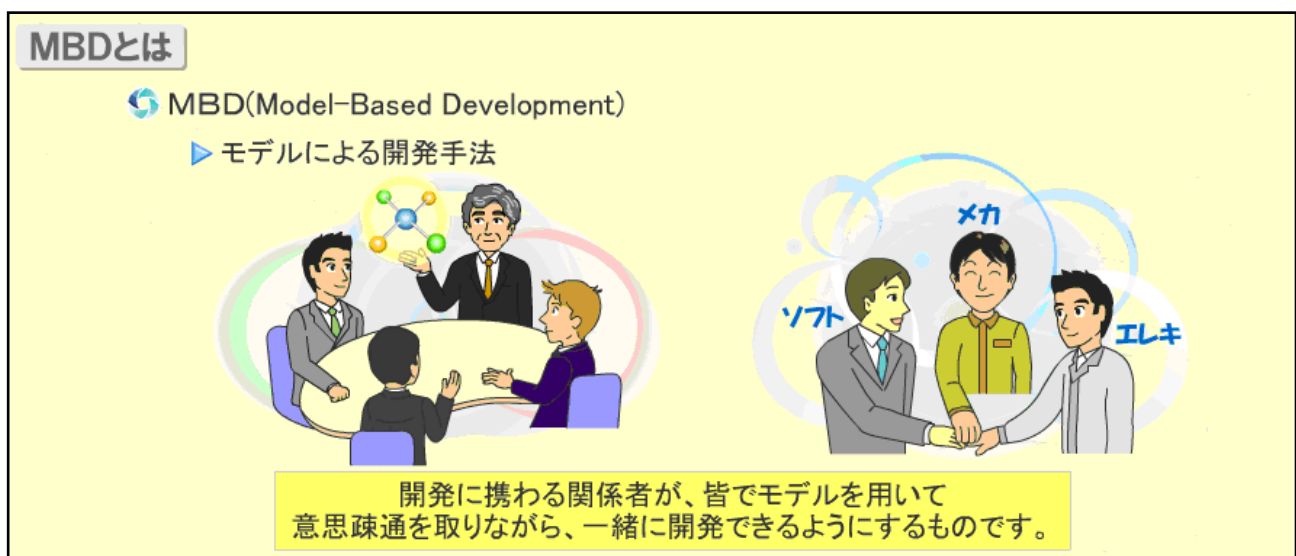


図 3-1 : MBD とはモデルによる開発手法

現在の開発でもモデルは用いられています。

しかし、開発に関わる全ての人々が共有できるものではありません。

MBD では、モデルを開発者全てが共有し、互いに分かり合った上で開発を行います。

製品開発では、様々な開発組織が関わっています。

開発に必要な部品を分担して生産していきませんが、作って欲しいものを誤解無く正確に伝えることは難しいことで、開発組織間の壁のようになっています。



図 3-2：開発組織間の壁

MBDでは、このコミュニケーションの壁を小さくし、互いの仕事を分かり合って開発が行えるようにしていきます。

MBD を表すキーワード

MBD を表すキーワードを4つ、紹介します。

1、見える！

機能をモデル化することで、属人化を抑え、機能のつながりを認識しやすくします。

2、動く！

シミュレーションでその仕様、機能がどのように動作するのか確認できるようにします。

3、分かり合う！

共通フレームワークの導入により、内容を誤解無く、解釈できるようにします。

4、自動化！

自動化技術（自動コード生成・自動検証）の導入により、機械的な作業を簡略化できるようにします。

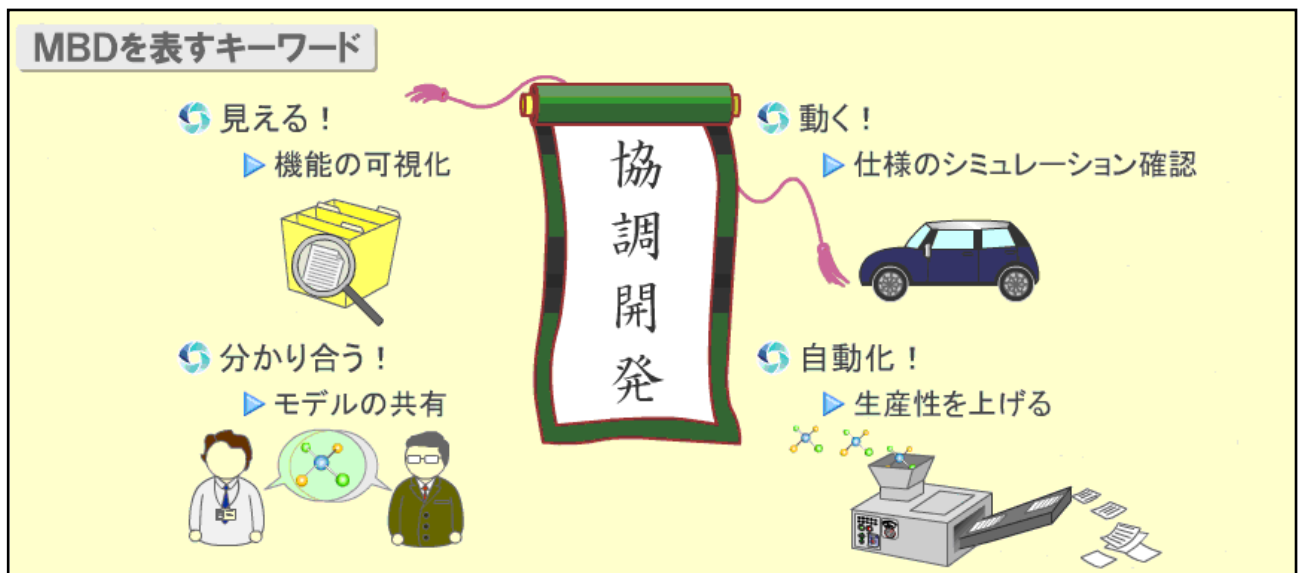


図 3-3 : MBD のキーワード

これら4つのキーワードより、

MBDでの開発は、協調開発であるということが出来ます。



MBDの定義

参考: MBDの定義 (JMAABでの定義)

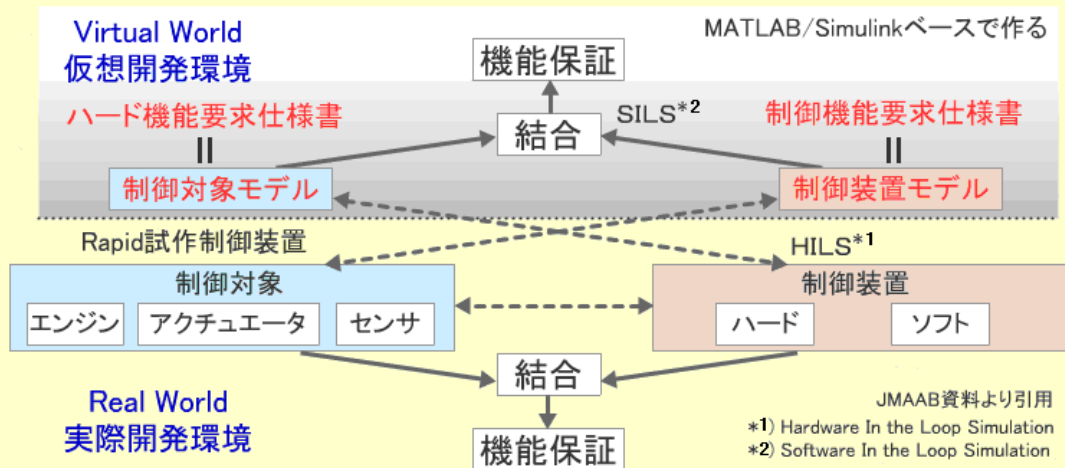
▶ MBD (Model-Based Development) とは？

- 複雑化・高度化した現代の自動車制御システム開発に於て、MATLAB/Simulink 等の CAE ツールによって制御装置と制御対象の機能をモデル化し、それらを実行可能な仕様書として用いることで、製品ライフサイクル全般に渡った品質向上と開発効率向上を目指した開発手法のことである。
- シミュレーション技術を駆使することで、高度な機能確認を実施でき、かつ、複雑な開発工程のルーチンワーク化を促進することで、自動化・省力化にも貢献する。

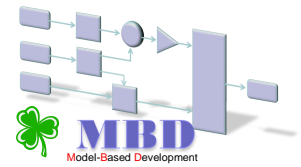
▶ モデルの定義:

- 対象の機能が図示されており、一意的に解釈できる物

制御システムにおけるモデルの位置づけ



4章 MBD の特徴



4章 1節 機能のモデル化

MBD・モデルベース開発の一番大きな特徴はその名の通り、「モデル（≡ 図）を基盤として開発を行う」ということです。

モデルとは、「誰でも一意的に解釈できるもの」（曖昧な表現ができず、理解に複雑なルールが不要）です。

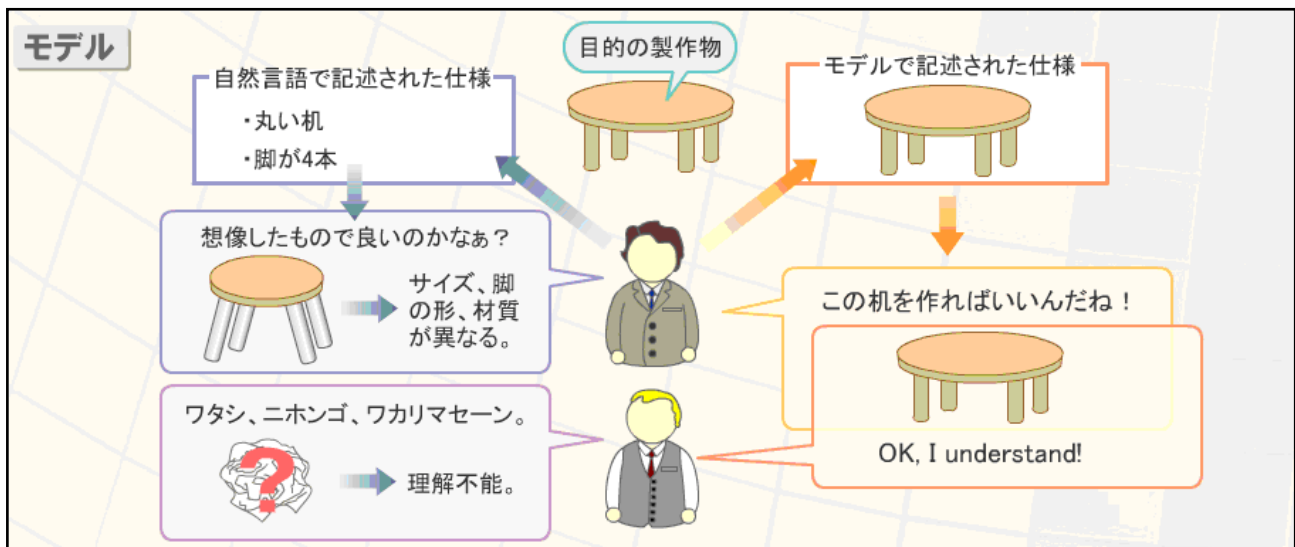


図 4-1：自然言語とモデルの比較

このモデルを使った開発手法である MBD においては、制御対象（ハードウェア）と制御装置（主にソフトウェア）の仕様書をモデルで作成し、開発メンバー間の「コミュニケーションツール」とすることで、全員に誤解の無い精度の高い意思疎通ができます。

4章2節 シミュレーション（実行可能な仕様書）

MBDにおける仕様書であるモデルが、従来の仕様書と大きく異なる点は、MBD ツールのシミュレーション機能により、動作（機能）の確認ができることです。

従来の仕様書の多くは、「文字ベース」+「図」でした。開発をするためにはこれらを読み、動作を想像して理解していました（文字ベースなので、記述漏れや動作の捉え間違いも多く発生します）。しかし、MBDのモデルによる仕様書であれば、シミュレーションを実行することで、その実際の動作（機能）を画面上で見られる（SimulinkのScope、Displayブロックなど）ので、仕様が明確に理解できます。

また、仕様作成者は、機能要求者とシミュレーションを使って要求の確認が可能です。要求者は、具体的且つ直感的に動作を見ることができ、理解が容易になります。

このモデル上でシミュレーションができるため、「実行可能な仕様書（動く仕様書）」（Executable Specification）と呼ばれています。

設計工程でのモデリング終了後に、即モデルシミュレーション検証が可能となるため、上流工程の時点で問題を早期発見し対処することができます。よって、下流工程以前に誤りの発見ができ、手戻りの増加工数を無くすることができます。

また、従来実機や試作品などハードウェアを用いていたテストを、ハードの代替となるモデル（プラントモデル）でシミュレーションすることで、実機・試作品の作成を少なくできます。

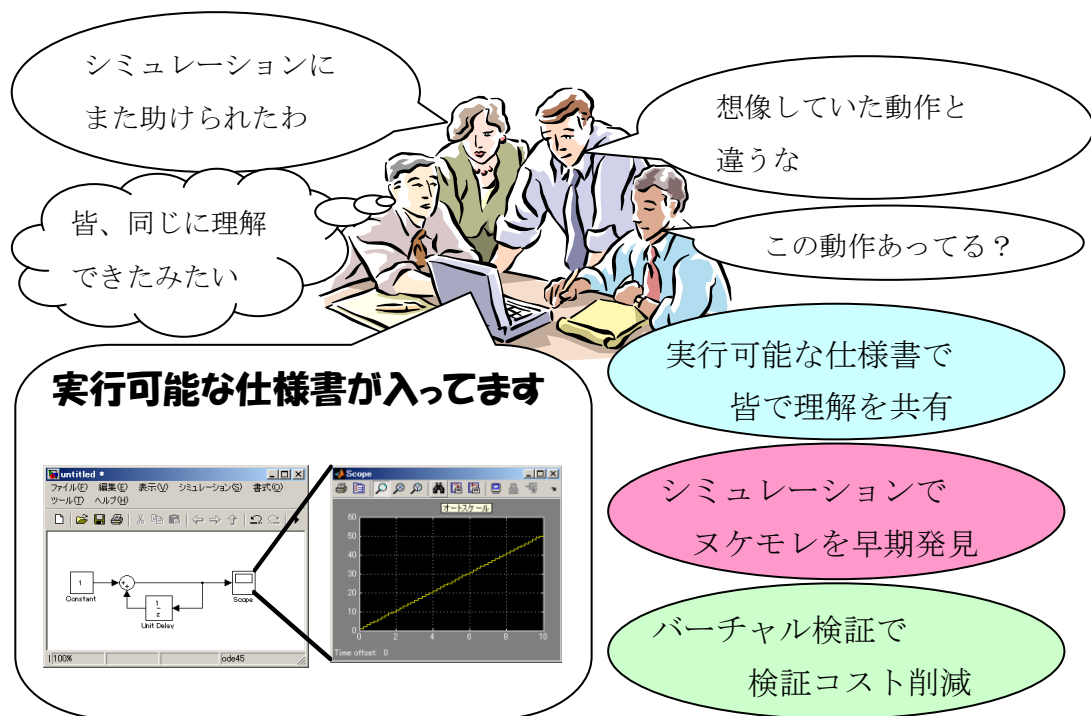


図 4-2：実行可能な仕様書の特徴

4章3節 自動コード生成

MBD ツールの大きな特徴として自動コード生成機能が挙げられます。

これは制御モデルを機能的に等価なプログラムコードに変換する機能です。(生成されたプログラムコードをマイコン命令に変換する(コンパイルする)ことで実機を動作させます。)

この機能を利用できるようなモデルを作成し、自動コード生成を行うことにより、従来プログラマによって行われていたコーディング作業を無くすことができます。それによるメリットは、以下の2つです。

- ① 今まで仕様書を満たすプログラムを書くいわば変換作業を人(プログラマ)の手により行ってきましたが、人間である以上ミスをすることがあります。またプログラマのスキルが不足している場合、作成されるコードの品質(不具合・可読性・実行効率など)が著しく低いものとなってしまいます。

MBD ツールでは、人為的ミスの無い、均一的な品質のコードが作成可能となります。



[memo]

この観点に限って言うと、MBDの自動コード生成機能はC言語などの高級言語のコンパイラと類似でかつ上位のものとも考えることもできます。

自動コード生成：モデル → 高級言語
コンパイラ ： 高級言語 → 実行モジュール(機械語)

高級言語が登場する前のソフトウェアはアセンブラ言語という、処理系に依存した機械語命令(バイナリコード)と1対1に対応したような言語でいわば直接プログラミングを行っていました。

高級言語の登場により、人間はより人間の思考に近い論理的で高度な命令を使用してプログラミングを行なえるようになりました。機械語への変換はコンパイラが行ないます。

現在では組み込みシステムでも、デバイスドライバやパフォーマンスが求められるクリティカルな部分などごく一部を除いて、ほとんどが高級言語に置き換わっています。

モデルは、高級言語よりも高級な、より人間の思考に近いプログラミングツールと考えることができます。

- ② 従来は、設計成果物（仕様書）と実装成果物（コード）が別のものとして、二元管理されている状態でした。そのため、実装後に仕様変更や追加に対応した際には、両者の同期が取られず、更新された仕様書が存在しない状況も見受けられました。

MBD ツールでは、コードはモデルから自動生成されたものを使うので、コードと仕様書（モデル）の内容が一致し、常に更新された仕様書（モデル）として一元管理することができます。



[memo]

修正要望が多発した場合、開発に余裕が無いとまず実装物作成を優先し、仕様書への反映は後回しになり、そのまま忘れ去られてしまいがちです。

この修正部分に更に手を加える場合、当時の経緯を知る人でなければ、急造された複雑なコードを解析してその仕様を確認するという非常に骨の折れる作業が必要となります。

このような状況に置かれた開発チームでは、得てして人の入れ替わりが激しいため、この作業の発生する可能性は非常に高く、そしてこの作業が更に開発を切迫していくのです。

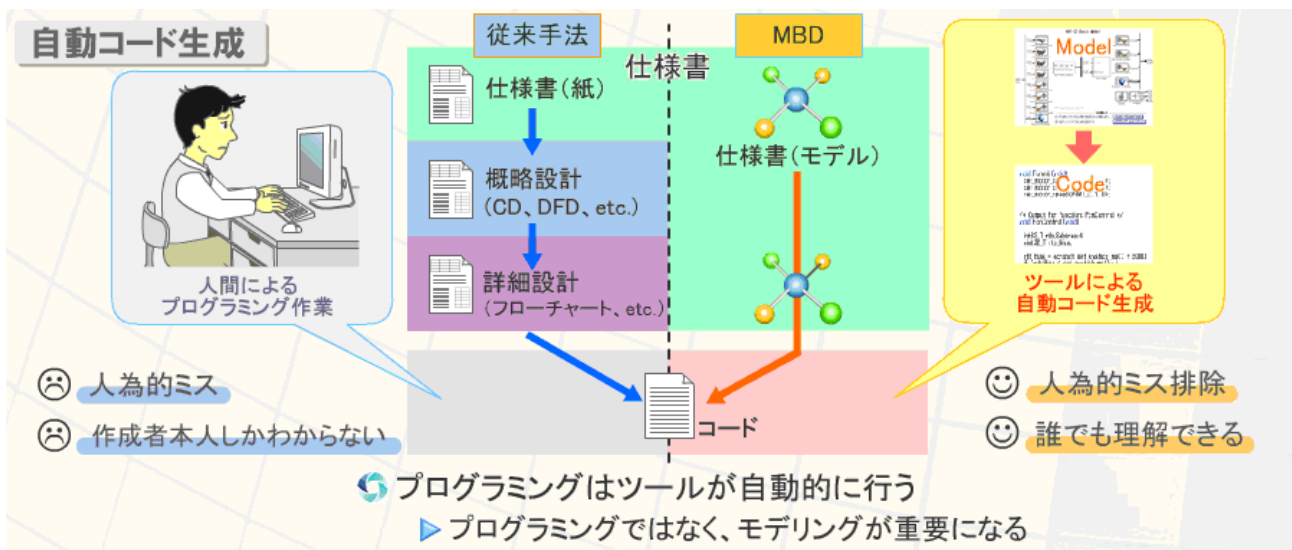


図 4-3 : 自動コード生成

4章4節 フロントローディング

・従来開発では試作で確認する

従来開発では試作したものが要求にかなっているかどうか確認するため、試作品を作り、動作の検証をします。そこで不具合が見つかった場合、原因を特定し、再び設計を行います。そのため大きな手戻りが発生します。

設計後、もう一度試作し、検証を行います。

手戻り・試作を繰り返していった場合、試作、検証、設計変更には膨大な時間とコストが発生します。

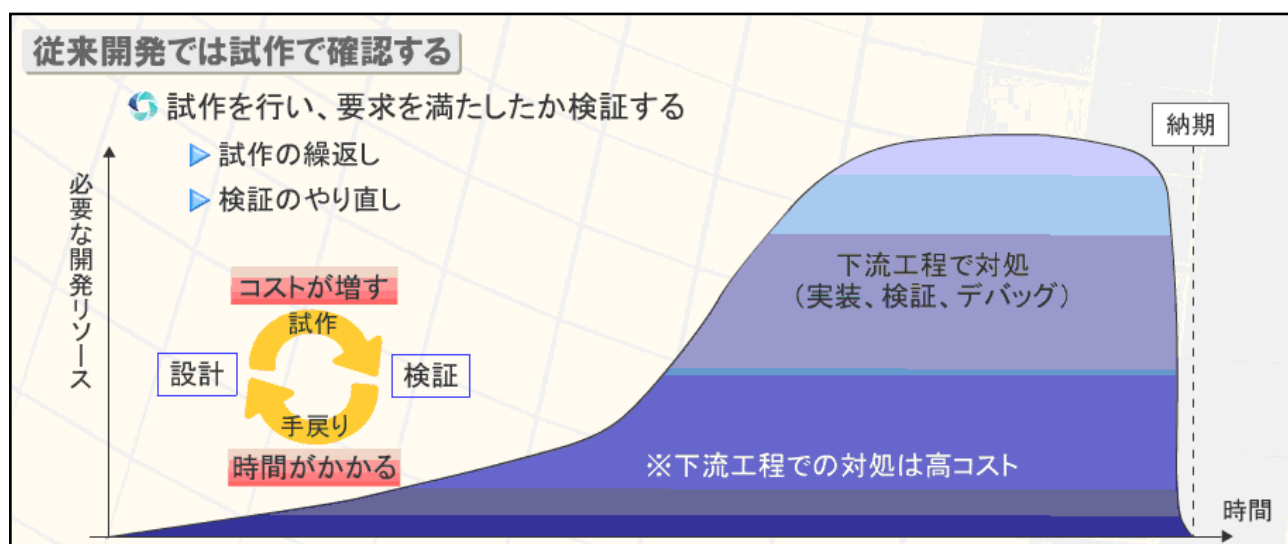


図 4-4 : 従来開発では試作で確認する

• MBD ではモデルで確認する

MBD では、試作品の代わりにモデルを使って動作の検証を行います。試作を待たずに結果を知ることができるため、時間を短縮することができます。また、試作回数を少なくできると考えられるため、その分のコストが減ります。

設計段階で CAE ツールなどを用いた十分なレビューを行い、不具合要因をできるだけ除いた設計書を作成し、それをもとに試作、検証工程に入る、というスタイルは、よく知られていると思います。

MBD は、これを開発プロセス全体に渡って行おう、という開発スタイルと言えます。

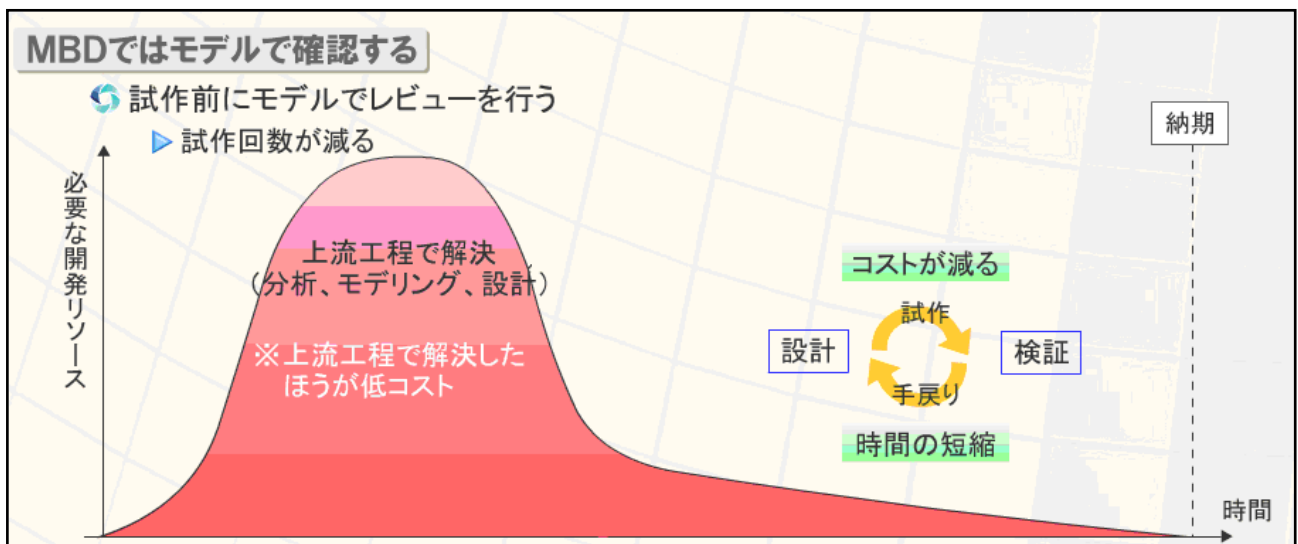


図 4-5 : MBD ではモデルで確認する

・開発全体のフロントローディング

MBD プロセスでは、開発プロセス全体をフロントローディングし、実行可能な仕様書のもと、コンカレント開発を行います。開発期間全体を短縮させ、製品のより早い市場投入を目指します。

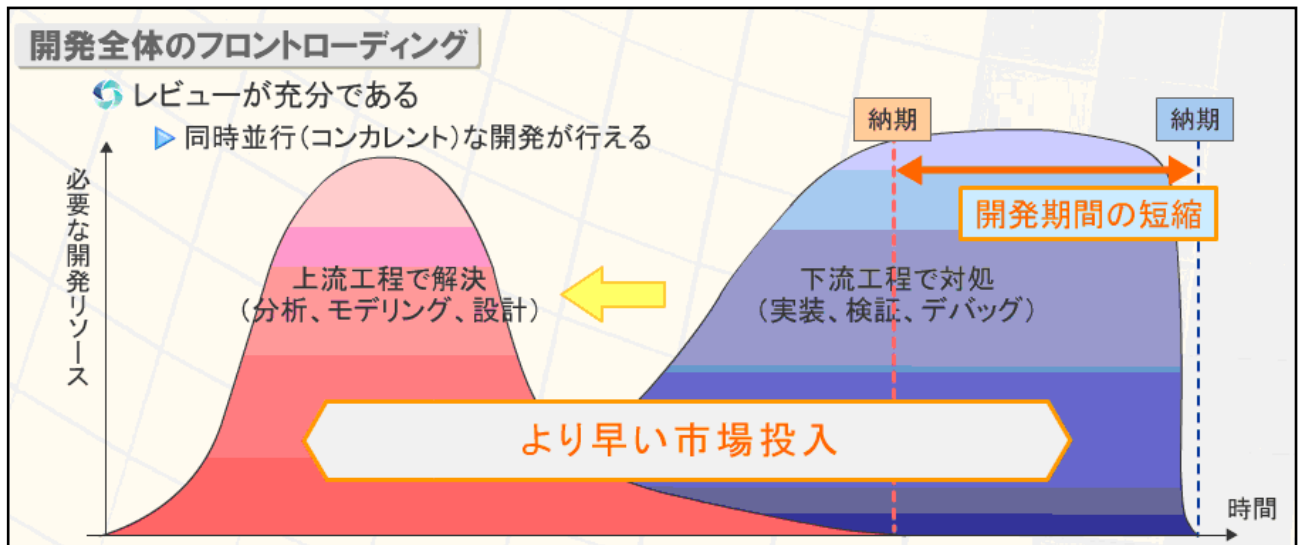


図 4-6 : 開発全体のフロントローディング

4章5節 コンカレント開発

これまでの開発では、1つの工程が終了するまで、次の工程を開始することができませんでした。MBD適用後では、モデリングフェーズで、モデルを使って実行可能な仕様書の作成を行います。

モデルを用いることで、企業間でのコミュニケーションを円滑にします。また、動くことが保証された仕様のもと各部品を並列して製造することができるので、その分全体の開発期間が短縮されます。

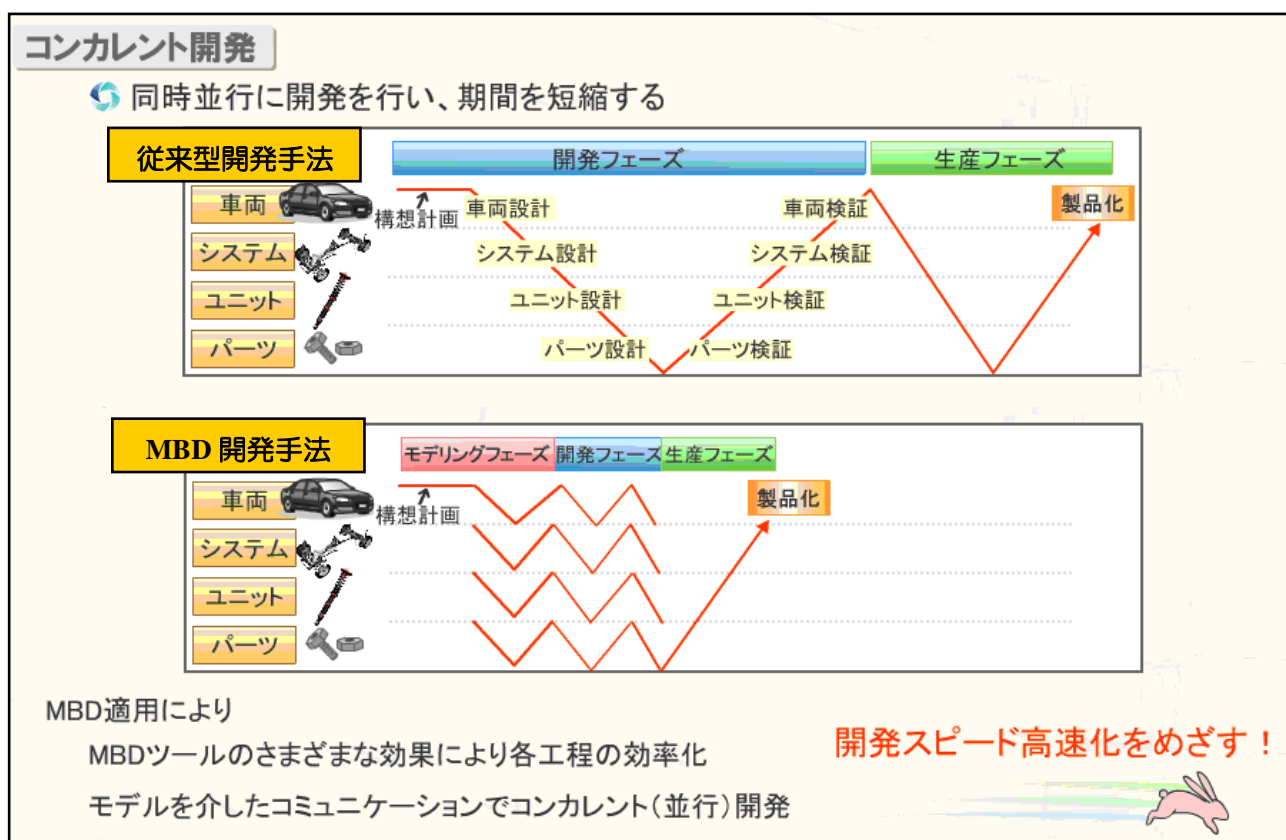


図 4-7：コンカレント開発

この章のまとめ

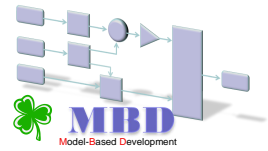
MBDでは、要求される機能をモデルで表すことで、わかりにくい機能をわかりやすくします。また、モデルをシミュレーションして動作を確認し、開発者間のコミュニケーションを円滑に進めます。さらに、自動コード生成機能を使用することで、プログラミング時間が短縮でき、また、モデルがドキュメントとなるため最新のドキュメントを一元管理できます。

これらの特徴は、複雑化・大規模化するシステムへの対応を可能とすることが見込まれます。

さらに、実行可能な仕様書を用いたコミュニケーションは、開発プロセスの姿を大きく変更します。プロセス全体をフロントローディングすることで、逐次的な開発を並列的にし、開発効率を向上させることが見込まれます。



図 4-8 : MBD の特徴のまとめ



5章 JMAAB

(Japan MATLAB Automotive Advisory Board)

5章1節 JMAAB とは、

JMAABは、国内自動車メーカーと同自動車用制御装置サプライヤーが参加しているMATLABのユーザー会です。

「開発環境構築は協調し、競争は製品で！優れた環境でレベルの高い競争をしよう！」をスローガンに、モデルベース開発の推進とMATLAB/Simulinkベースでの設計・開発環境の発展、自動車メーカーとサプライヤーの境界を越えた効率的な開発環境を実現させるために、様々な活動を行っています。

JMAABとは

JMAAB (Japan MATLAB Automotive Advisory Board)
国内自動車メーカーと、同自動車用制御装置サプライヤーの
MATLABユーザー会

活動目的

- モデルベース開発 (MBD) の推進、MBDプロセスの早期実現
- MATLAB/Simulinkベースでの設計・開発環境の発展
- 自動車メーカーとサプライヤーの境界を越えた効率的な開発環境の実現

スローガン

- 開発環境構築は協調し、競争は製品で！
- 優れた環境でレベルの高い競争をしよう！

図 5-1 : JMAAB について

JMAABは、ユーザー会の発起人中心で構成されるボードメンバ、実際の活動を行うワーキンググループ (以下、WGと略称)、WG活動に参加しているコアメンバ、JMAAB活動に関心を持ちインターネットを通して会員登録した一般メンバで構成されています。

5章2節 MBD技術者の育成

MBD技術者育成の重要性

モデルベース開発の推進にあたり、CAEツールや設計技法の開発改良が進む中で、自動車制御系開発における各種ツールの知識や使いこなし、制御システムの設計検証技法、設計プロセスにおける管理ツールの活用手法など、自動車業界のエンジニアにとって必要となる知識やスキルの範囲が広がり、何らかの指針が必要になってきています。

優れたツールも、最新の技術や手法を十分に取り扱える人材がいなければ成果を出すことができません。そのためMBD技術者育成は重要な、また急務な課題となっています。

MBDエンジニア育成WG

MBD技術者育成を推進するために「JMAAB MBDエンジニア育成WG」というワーキンググループ（以下、育成WGと略称）を2005年に発足しました。

育成WGのねらいは、自動車メーカーとサプライヤーの視点から、MBDにおけるモデリングやツールに関する知識・設計プロセスにおける活用手法など、マネージャ／技術リーダー／実務担当者を対象とした共通課題を明らかにして、MBDエンジニアに必要な教育プログラムの企画、及び共通指標となる認定レベルの制定を行うことです。

育成WGでは、IPA/SEC発行の「組込みスキル標準ETSS」をもとに、自動車制御系開発におけるMBDエンジニアを対象としたスキル基準・キャリア基準としてETSS-JMAABを作成し、自動車分野での人材育成に適用することを目的としています。

出典：『SEC journal №13 自動車分野のMBD技術者に必要なスキル』より引用

※ETSS-JMAAB文書は、ホームページ(<http://jmaab.mathworks.jp/>)で一般公開しています。

5章3節 ETSS-JMAAB

ETSS-JMAABについて

ETSS-JMAABは、自動車制御系開発におけるMBDエンジニアを対象として、開発技術、要素技術、管理技術に関するスキルとキャリアをまとめたものであり、以下の領域での活用を期待しています。

- ①自部署のスキルレベル把握と人材育成計画の立案
- ②外注者のスキルレベル把握（受け入れレベル規定等）
- ③教育カリキュラム構築 …他

各々の要素は、図5-2 に示すようにマッピングして関連付けています。

スキル基準 : MBDによる制御系開発に必要なスキルを体系的に整理するフレームワーク

キャリア基準 : MBDによる制御系開発に関わる職種名称や職掌とそれに求められるスキルを定義するフレームワーク

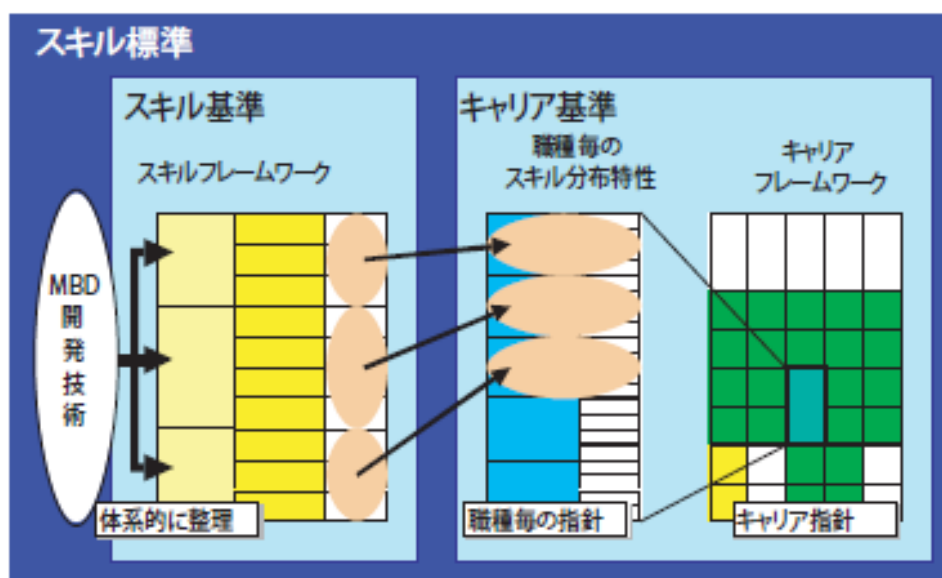


図 5-2 : ETSS-JMAAB の全体構成

スキル基準について

スキル基準は、MBDによる制御系開発に必要なスキルを明確化・体系化したものであり、人材育成・活用に有用な「ものさし」（共通基準）を提供します。スキルは「技術要素」「開発技術」「管理技術」のカテゴリで整理・階層化されています。図5-3にETSS-JMAABのスキルフレームワークを示します。

スキル カテゴリ ↓	スキル粒度				スキルレベル					
	第1階層	第2階層	第3階層(例)	説明	LV#0 初心	LV#1 初級	LV#2 中級	LV#3 上級	LV#4 最上級	LV#5
技術要素										
開発技術										
管理技術										

図 5-3 : ETSS-JMAAB のスキルフレームワーク

ETSSと同様に、スキルとは作業の遂行能力を指し、「～ができること」を表現するものであり、知識を有するだけではスキルとしては扱いません。知識は、スキルを発揮するために必要な構成要素で、特に自動車の製品知識と工学基礎知識は、MBDエンジニアに必要な知識となります。

スキル基準で定義する技術の範囲は、共通的に利用されるものを想定し、各企業や応用ドメインで利用される特有の技術に関しては扱っていません。また、スキル粒度の第3階層は、例にとどめています。運用する各企業で具体化・追加等が必要となります。

技術要素と開発技術カテゴリについては、図5-4、図5-5 でETSSとの違いを示します。



図 5-4：技術要素スキルカテゴリの第一階層

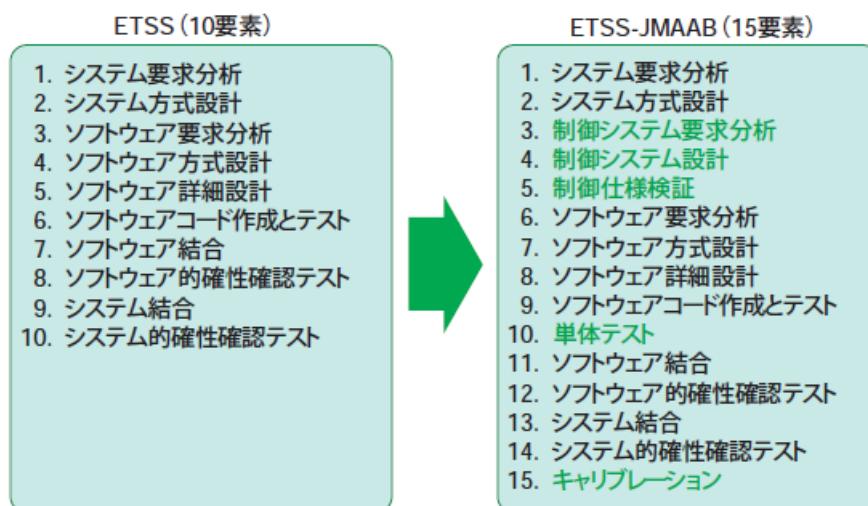


図 5-5：開発要素スキルカテゴリの第一階層

スキルレベルは、以下の0～5の6段階で定義しています。

- ・レベル0： 内容を知らない。
- ・レベル1：初心 内容を知っている。
- ・レベル2：初級 支援のもとに作業を遂行できる。
- ・レベル3：中級 自立的に作業を遂行できる。
- ・レベル4：上級 作業を分析し改善・改良できる。
- ・レベル5：最上級 新たな技術を開発できる。

キャリア基準について

キャリア基準は、MBDに従事する技術者の主な職種、その内容、レベル、求められるスキルを明示したものです。MBDによる制御系開発に従事する技術者の職種を14種類定義しています（図5-6）。

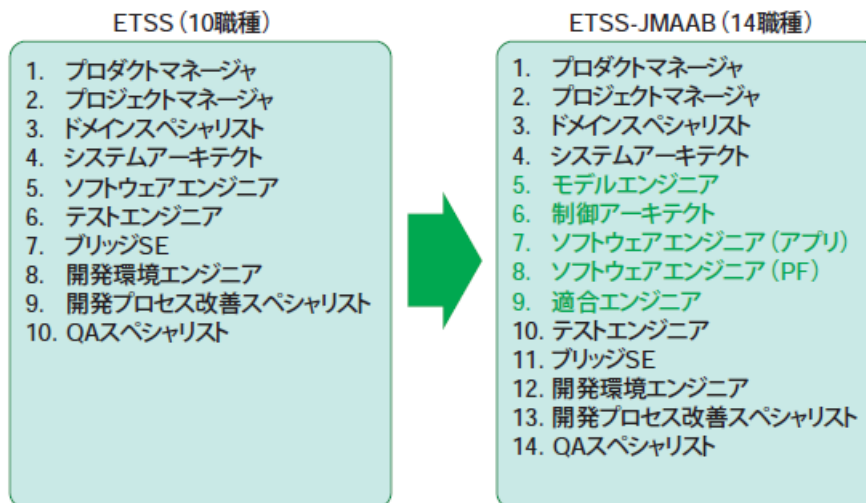


図 5-6：職種

キャリアレベルは、モデルベース開発の各職種において、技術者が専門性を持った人材として、価値創出に応じたスキルの度合いを表したもので、3段階のレベルで定義しています（図5-7）。

		エントリレベル	ミドルレベル	ハイレベル
要求作業 (役割) の達成	価値創造 への貢献			社内をリードする
			経験を知識化し、業務の改善や後進育成の面で応用できる	
			独力ですべてできる	
		指導の下でできる		

図 5-7：ETSS-JMAAB のキャリアレベル

制御システム設計に直接携わるエンジニア（ドメインスペシャリスト、システムアーキテクト、モデルエンジニア、制御アーキテクト、ソフトウェアエンジニア、適合エンジニア、テストエンジニア）8職種については、具体的なスキルレベルマップを示しています。

ETSS-JMAAB との対応

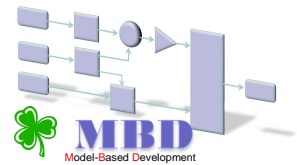
本教科書は、「MBD エンジニアスキル標準」の開発技術要素に当てはめると、設計からテスト工程の範囲について説明しています。

本教科書での学習後の成果目標は、スキルレベルを初心レベルに向上させることとなります。

スキルカテゴリ		教育レベル					
		未経験	初心	初級	中級	上級	最上級
開発技術	システム要求分析						
	システム設計						
	制御システム要求分析						
	制御システム設計						
	ソフトウェア要求分析						
	ソフトウェア設計						
	ソフトウェア詳細設計						
	ソフトウェアコード作成とテスト						
	ソフトウェア結合						
	ソフトウェア適格性確認テスト						
	システム結合						
	システム適格性確認テスト						
	キャリブレーション						

設計～テスト工程の範囲について初心レベル(内容を知っている)に向上させる

図 5-8 : 成果目標



6章1節 MATLAB

本書では、MATLAB をモデルベース開発の共通プラットフォームとして使用します。MATLAB は、豊富な科学、技術計算のライブラリがあり、広く一般科学、応用科学、工学の幅広い分野で用いられています。

適用分野：

データ解析、実験・計測、制御システム、通信システム、信号処理、画像処理など

産業分野：

自動車、電機、航空宇宙、通信、環境/エネルギー、教育、医療/科学など

Simulink は、モデルベース開発のためのプラットフォームとして、ブロック線図環境、モデリング・シミュレーションによる設計環境、自動コード生成環境を提供します。

MATLAB/Simulink には、豊富なツール群（拡張ライブラリ/追加オプション）があり、統一された環境におけるシステム開発を可能としています。

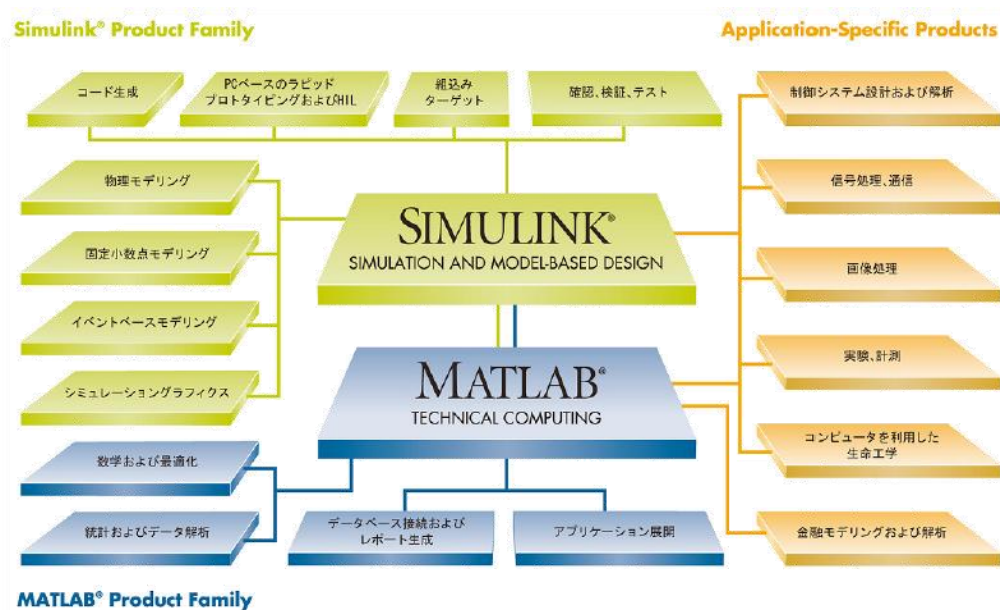


図 6-1-1 : MATLAB/Simulink プロダクトファミリー

出典：<http://www.mathworks.co.jp/products/pfo/>

開発プロセスに対応した、MATLAB/Simulink プロダクトを図 6-1-2 に示します。

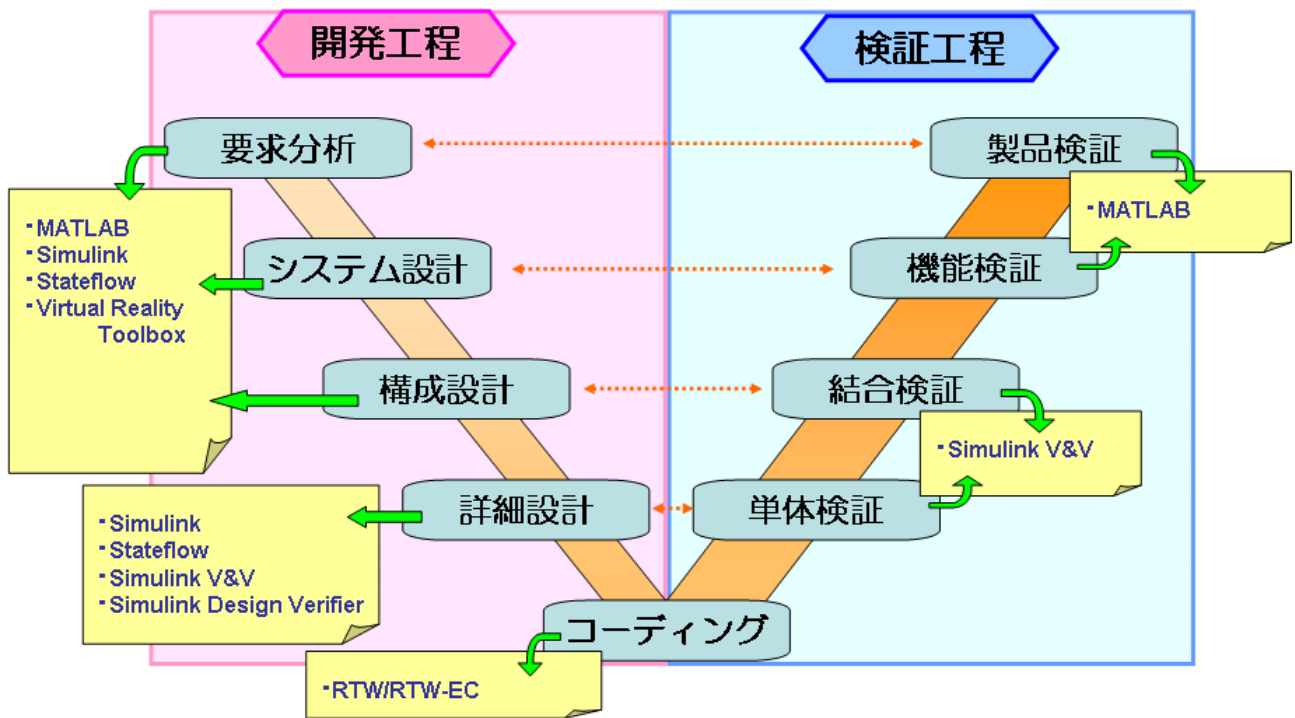


図 6-1-2 : 開発プロセスと MATLAB 製品

本書では、MATLAB 及び Simulink を実習環境として使用します。

※MATLAB 及び Simulink は米国 The MathWorks 社の登録商標です。

6章2節 MBD ツールに触れてみよう。

6. 2. 1. モデルを描いてみよう（前準備）

MATLAB/Simulink は、ワープロ、表計算ソフト等の図形描画操作に近い操作方法となっています。

- MATLAB を起動してみよう。

デスクトップ上の「MATLAB R2007b」を実行すると「MATLAB R2007b」のウィンドウが開きます。スタートメニューからでも同様に起動できます。

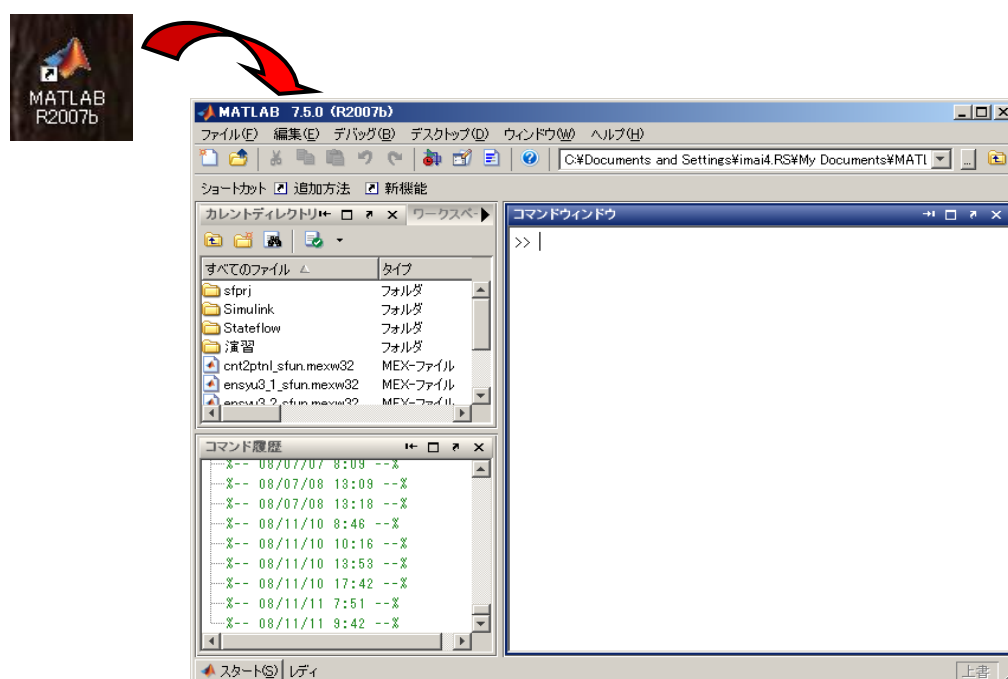


図 6-2-1 : MATLAB 起動

• Simulink を起動してみよう。

ツールバーの「Simulink」アイコンをクリックすると「Simulink Library Browser」が起動します。コマンドウィンドウに「simulink」と入力し、改行(Enter キー)しても同様に起動します。

※Simulink は「モデリング」、「シミュレーション」を行うツールです。

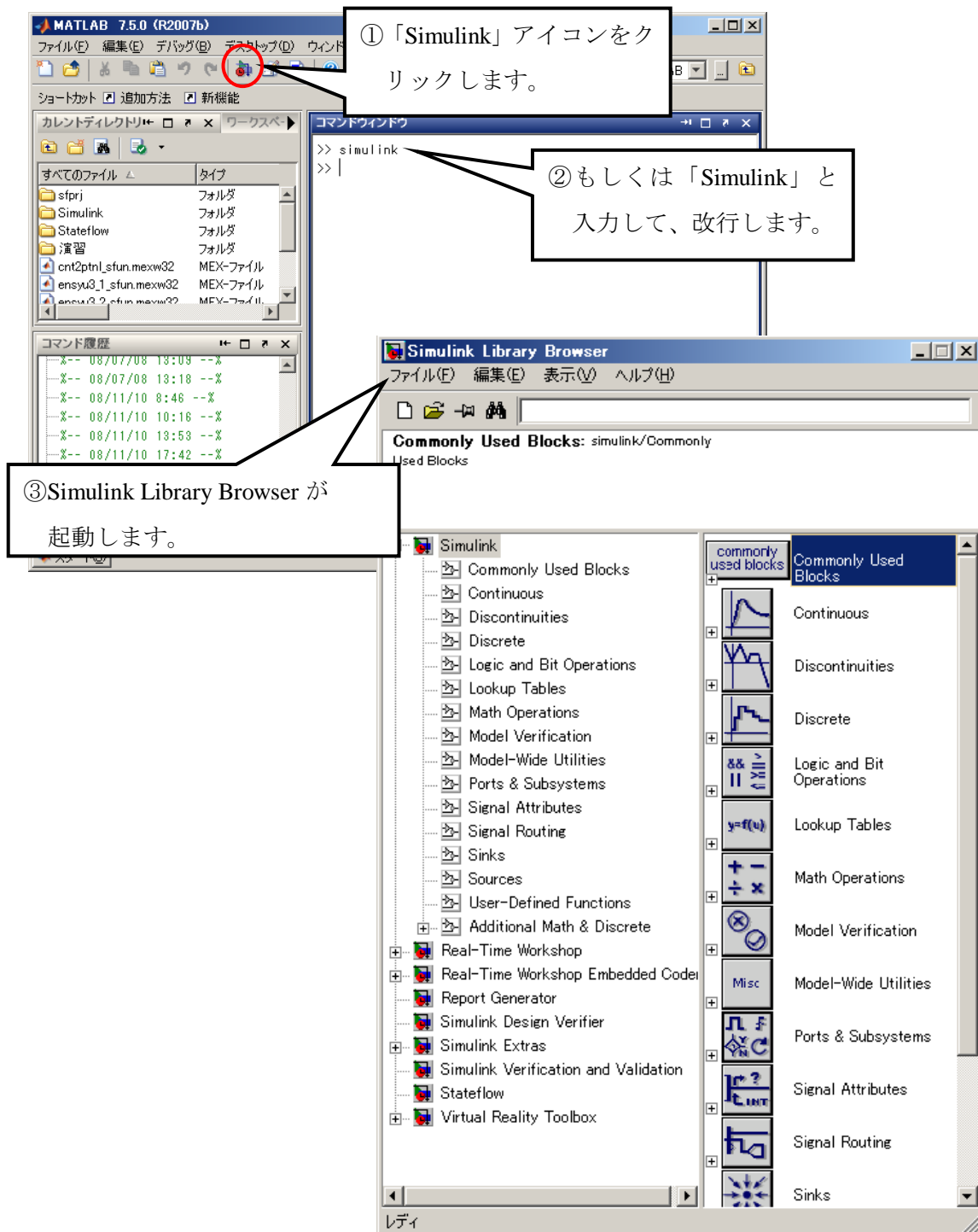


図 6-2-2 : Simulink 起動の流れ

- モデルを描くキャンバスを用意しよう。

「新規モデルの作成」を実行するとメニューしかない別ウィンドウでキャンバスが開きます。

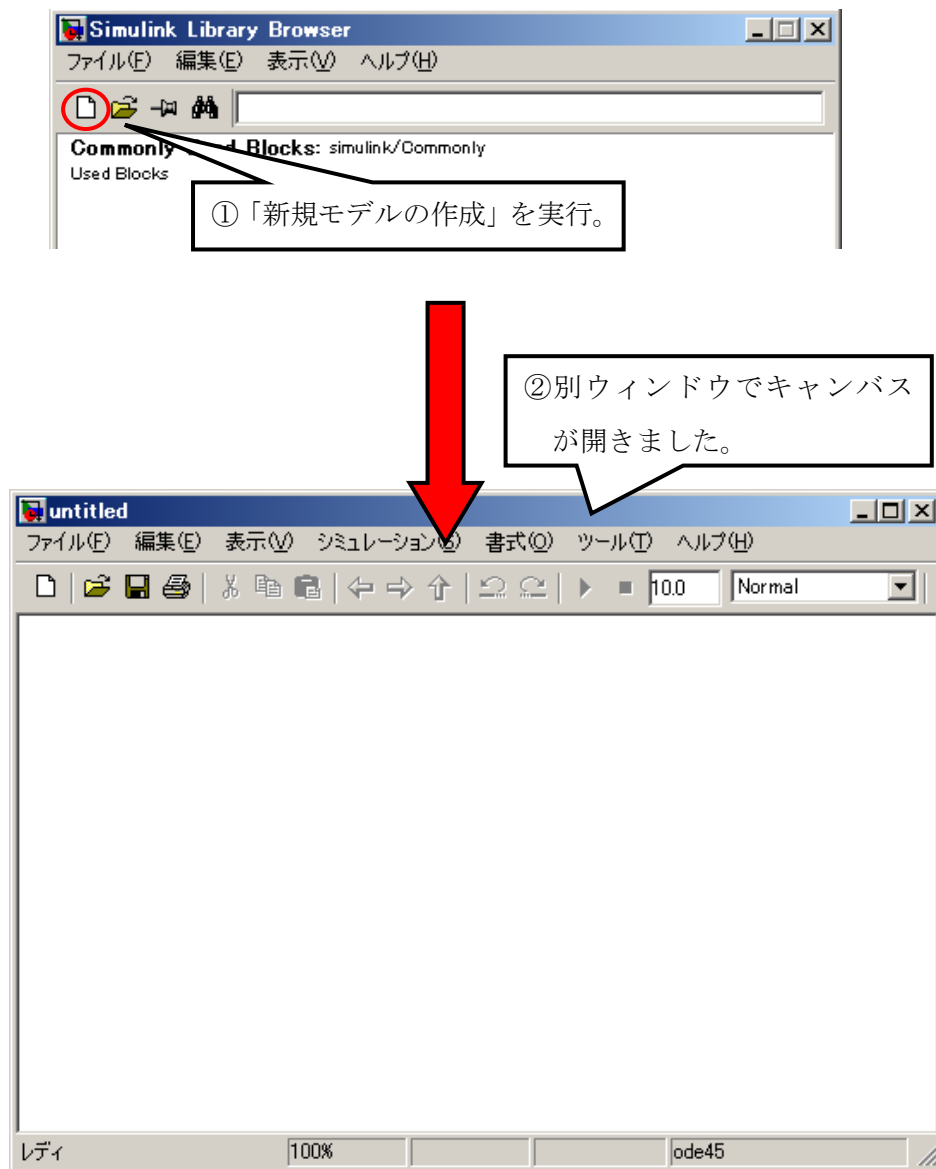


図 6-2-3 : キャンバス作成

6. 2. 2. モデルを描いてみよう (ブロックの描画)

- 描画するブロックを選択しよう。(ブロックの選択)

「Simulink Library Browser」の「Simulink ライブラリ」の「Math Operations」を選択します。

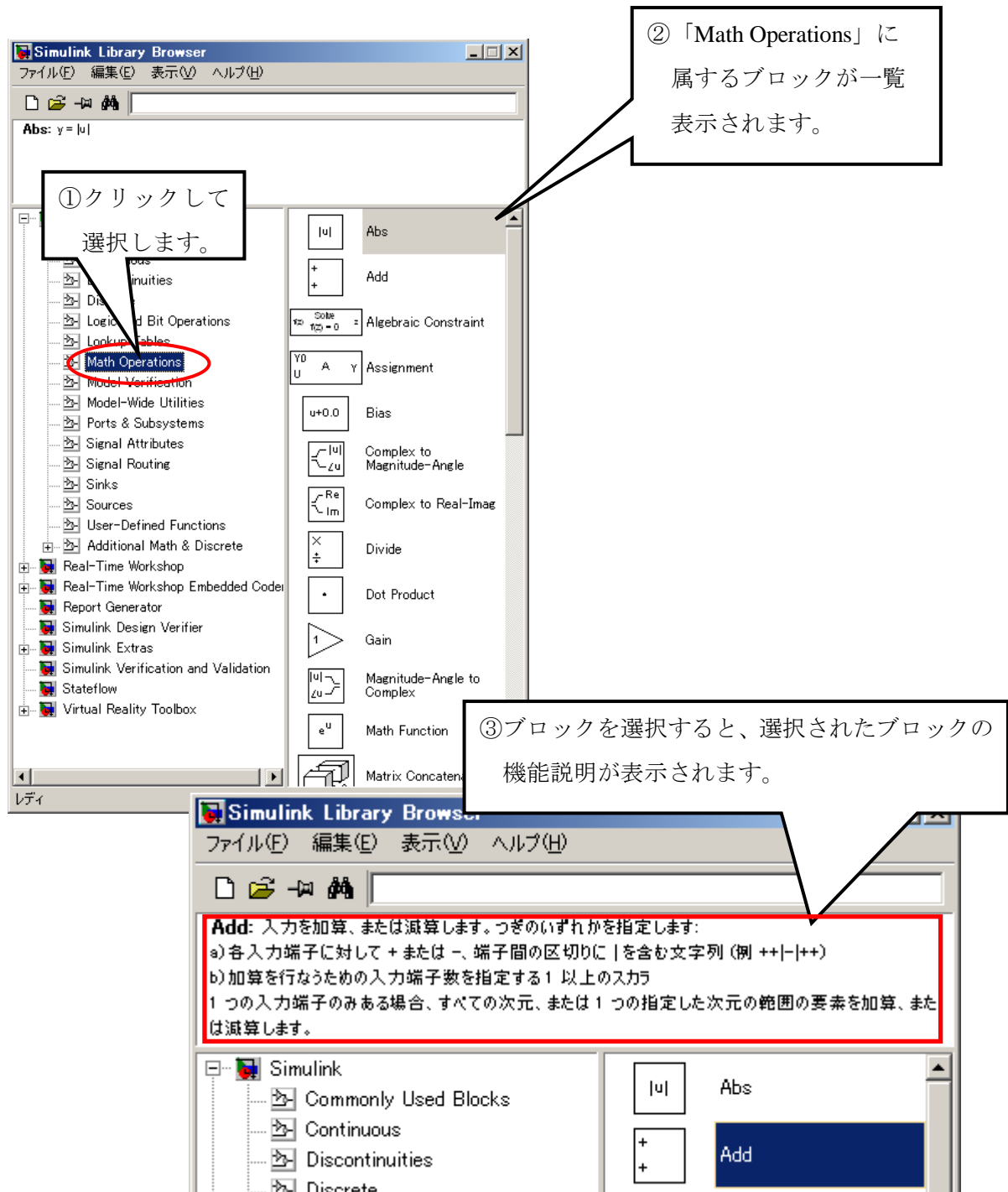


図 6-2-4 : ブロックの選択

- キャンバスにブロックを配置しよう。(ブロックの描画)

選択した「Add」ブロックをドラッグして、キャンバス上でドロップするとキャンバス上に「Add」ブロックが配置されます。

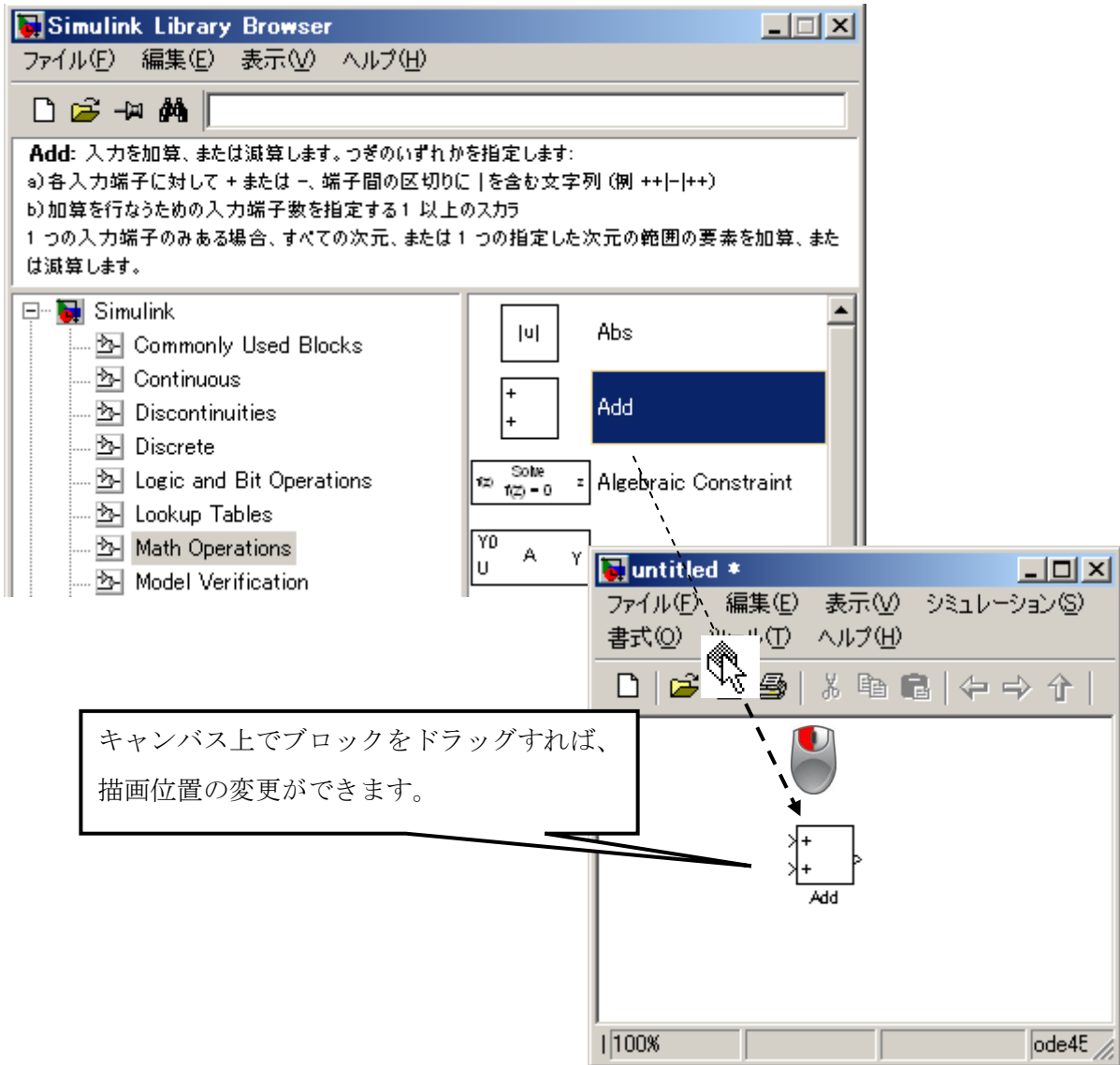


図 6-2-5 : ブロックの配置

同様に、「Sources」の中の「Constant」と「Sinks」の中の「Display」をキャンバス上に配置します。

※基本的にブロック線図は左から右へ信号（データ）が流れるように描くことが推奨されています。これに合わせてブロックを配置します。

- キャンバス上に既にあるブロックをコピーしよう。

キャンバス上の「Constant」ブロックを右ドラッグし、配置したい場所でドロップします。

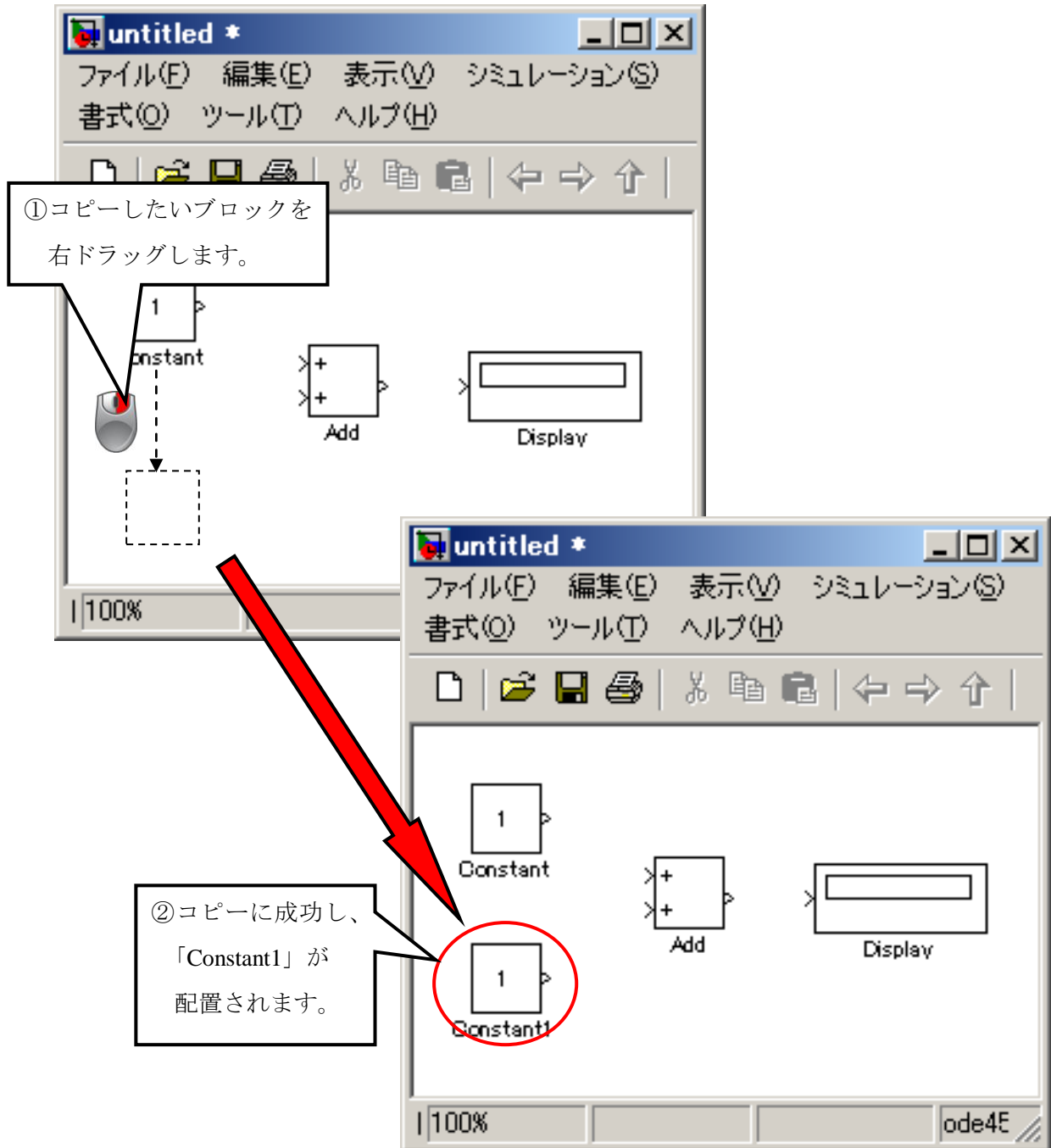


図 6-2-6 : ブロックの選択

6. 2. 3. モデルを描いてみよう（ブロック間の結線）

• ブロック同士を結線しよう。（ブロック間の結線）

「Constant」をクリックして、選択した状態にして「Ctrl」キーを押しながら、「Add」ブロックをクリックします。

これで「Constant」ブロックの出力端子と「Add」ブロックの入力端子が結線されます。

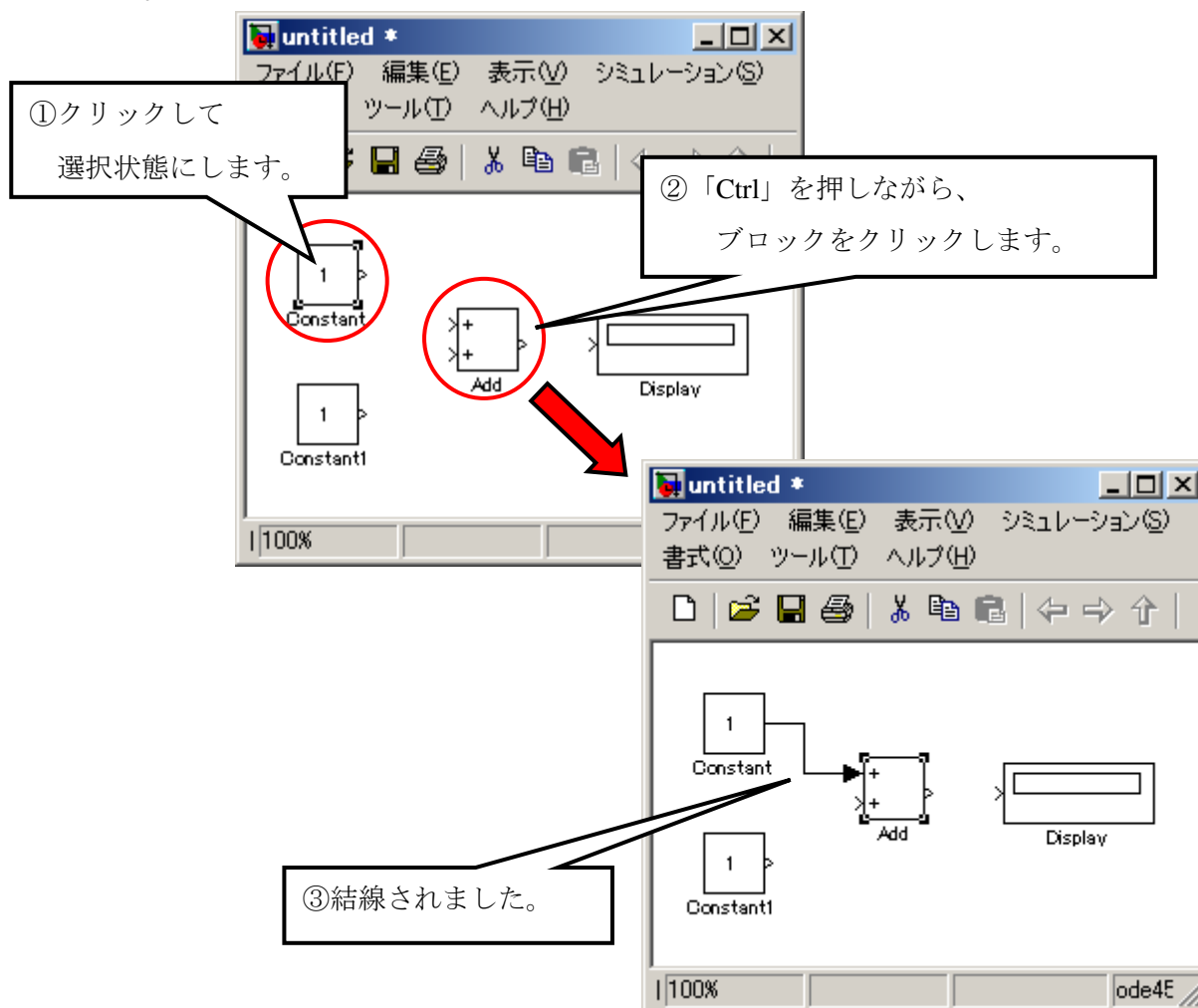


図 6-2-7 : ブロック同士の結線

同様に、「Constant1」と「Add」、「Add」と「Display」も結線して下さい。
結線はドラッグすることにより、位置を変更できます。また、ブロックを移動させると、結線も追従します。

※結線同士が交差すると、見づらく誤認識する可能性が高くなるので、可能な限り交差しないように結線することが推奨されています。

• 結線を削除するには？

削除したい結線をクリックで選択し、「Delete」キーを押すか、右クリックで表示されるポップアップメニューから「削除」を選択する。

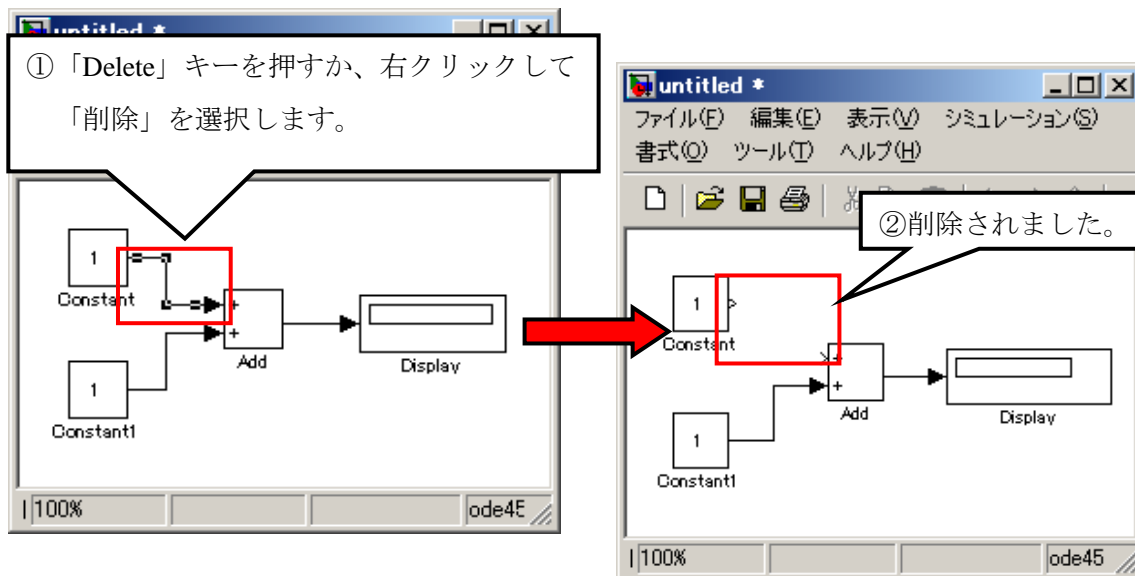


図 6-2-8 : 結線の削除

• 結線を分岐させるには？

一つの信号（データ）を複数のブロックへ供給する場合は、一つ目は1の方法で結線し、2箇所目からは既存の結線から分岐し供給します。分岐したい結線の分岐個所で右ドラッグして、結線したいブロックにドロップすることで、分岐結線できます。

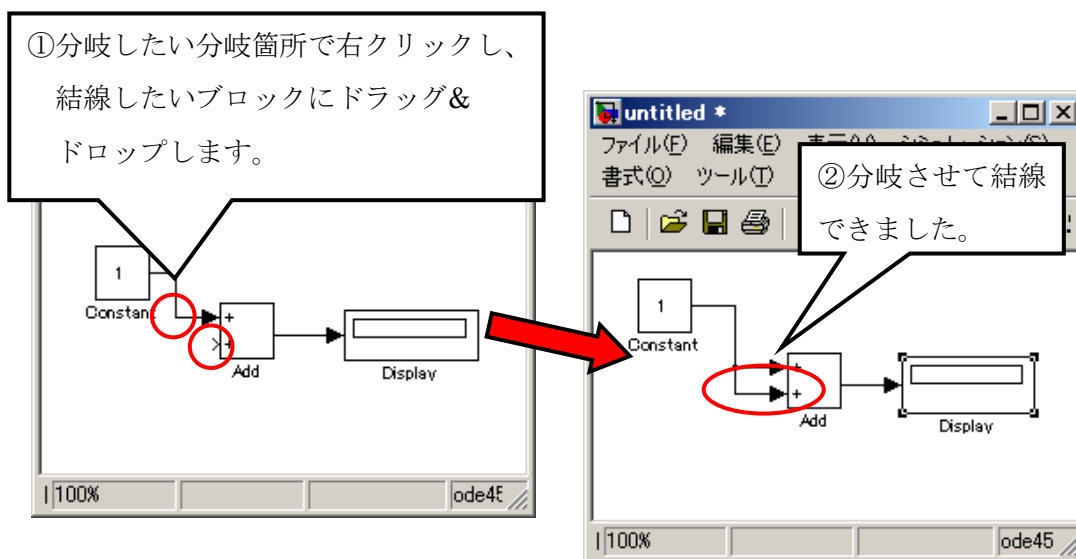


図 6-2-9 : 結線の削除

6章3節 シミュレーションしてみよう

6. 3. 1. モデルを動かしてみよう（シミュレーション）

- ・シミュレーションを実行します。

ツールバーの▶アイコンをクリックします。メニューの「シミュレーション」の「スタート」をクリックしても実行できます。

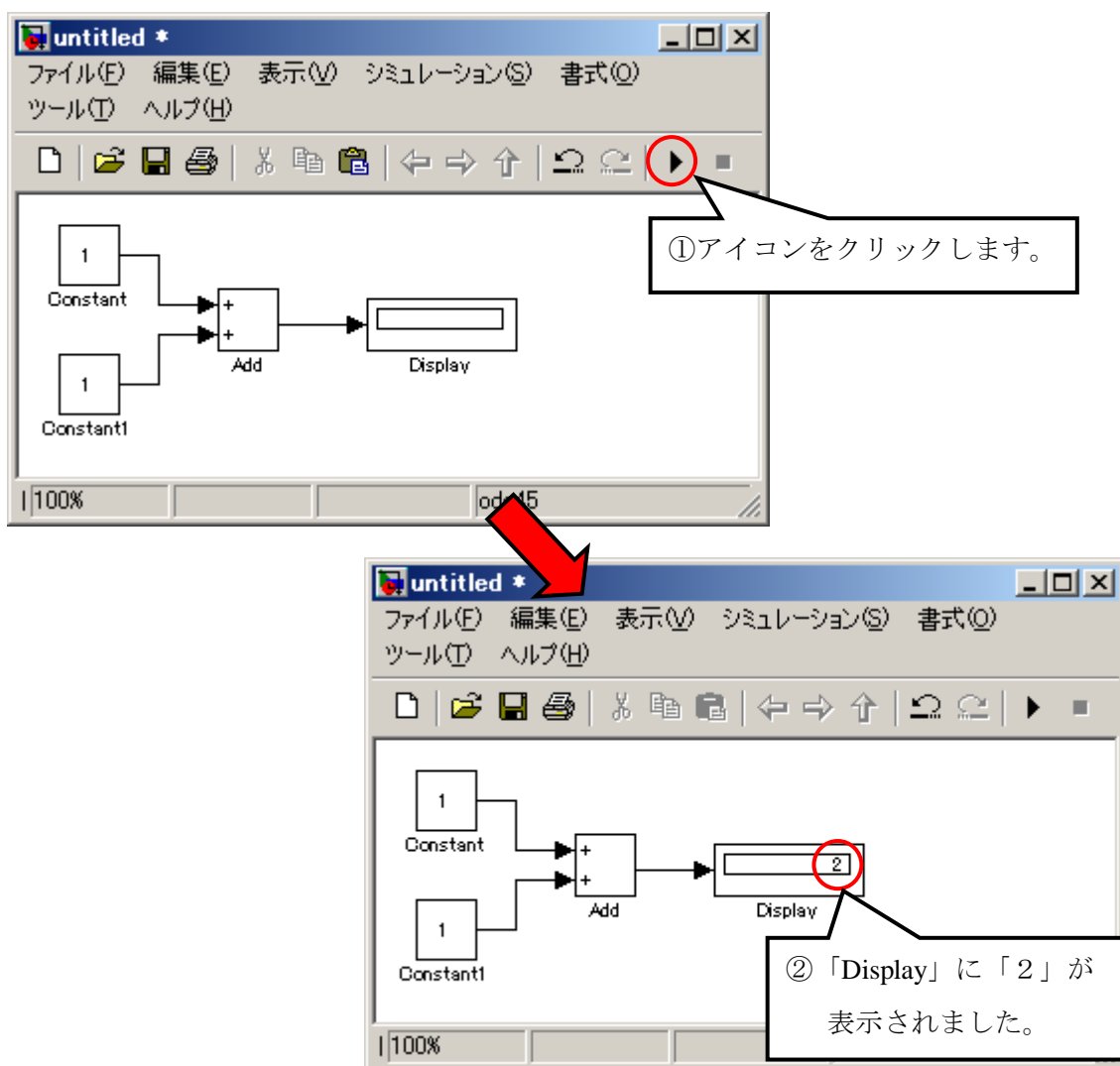


図 6-3-1 : シミュレーション結果

ここで問題です。

今、皆さんが描いたモデルは何を表現したモデルでしょうか？
推測してみてください。

• パラメータを変更しよう。

「Constant」ブロックをダブルクリックします。すると、「Source Block Parameters : Constant」というウィンドウが開きます。メインタブに定数という項目があり、「1」が設定してあります。この「1」を「2」に変更し、「OK」をクリックします。

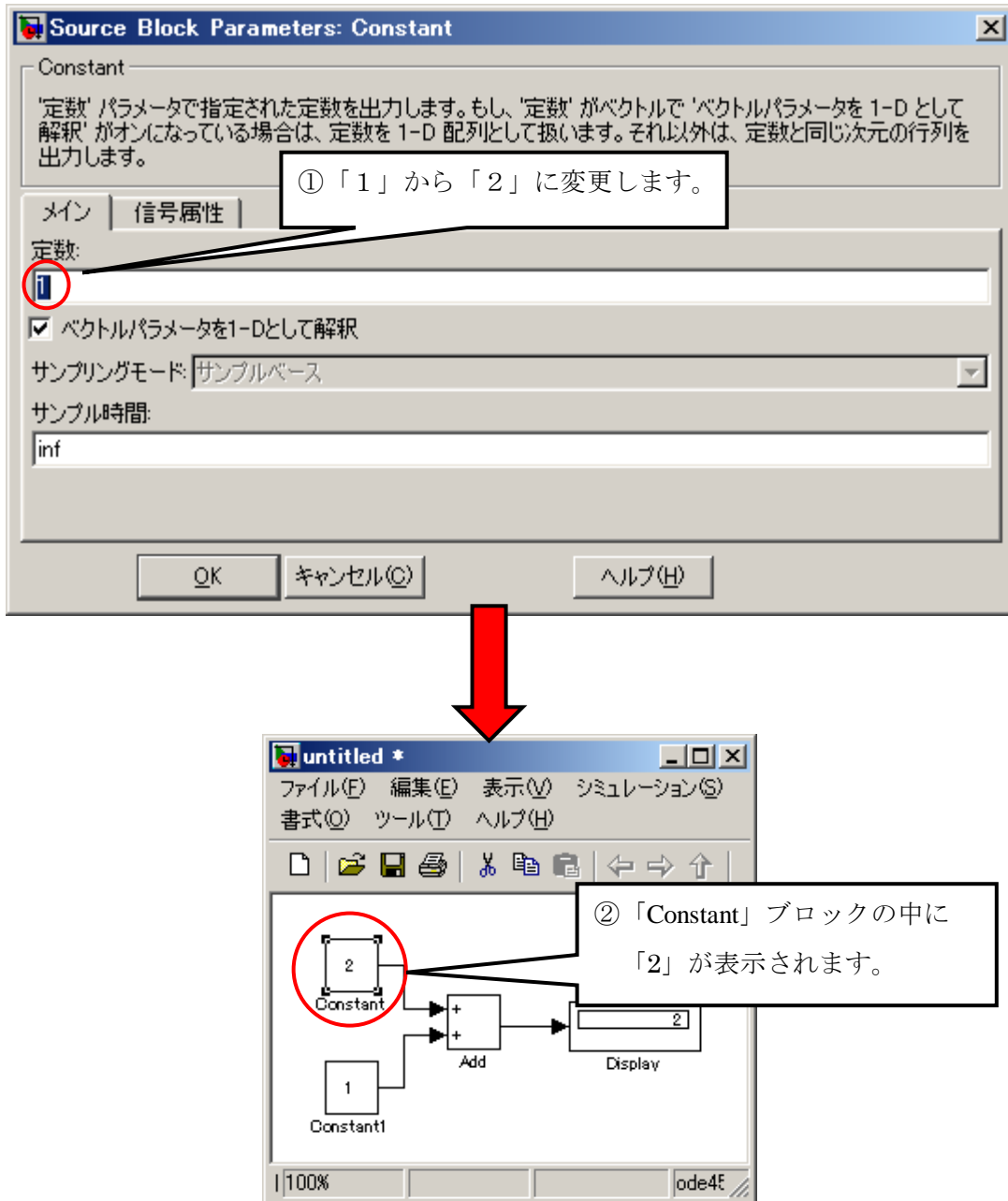


図 6-3-2 : パラメータの変更

- 変更結果を確認しよう。

再度、メニューの「シミュレーション」の「スタート」をクリックして下さい。

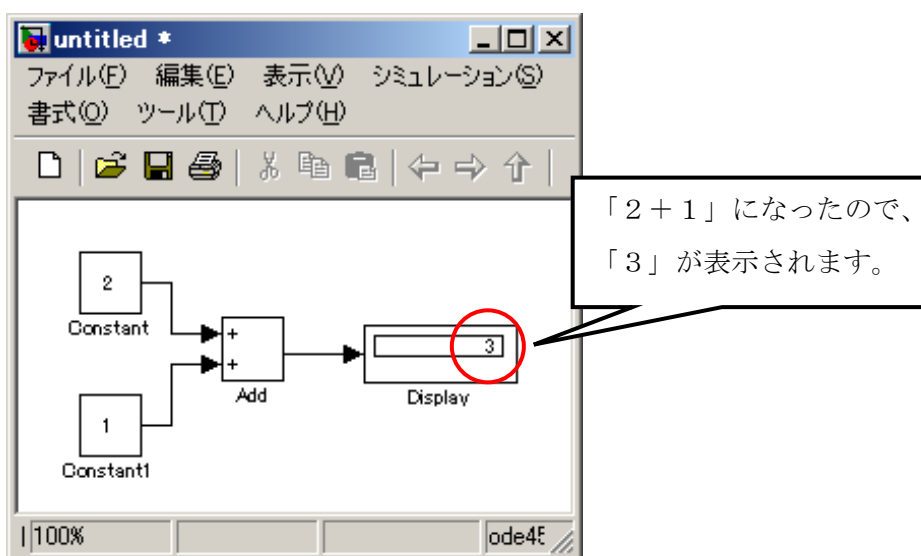


図 6-3-3 : パラメータの変更後のシミュレーション結果

「Constant1」、「Add」、「Display」ブロックもダブルクリックして、パラメータを比較しましょう。

※ブロックの種類ごとに持っているパラメータは異なります。

6. 3. 2. ブロックを変更し、動作を確認しよう。

• ブロックを変更しよう。

「Add」ブロックを削除します。すると「Add」ブロックが消え、「Add」ブロックに接続していた結線が赤い破線になります。

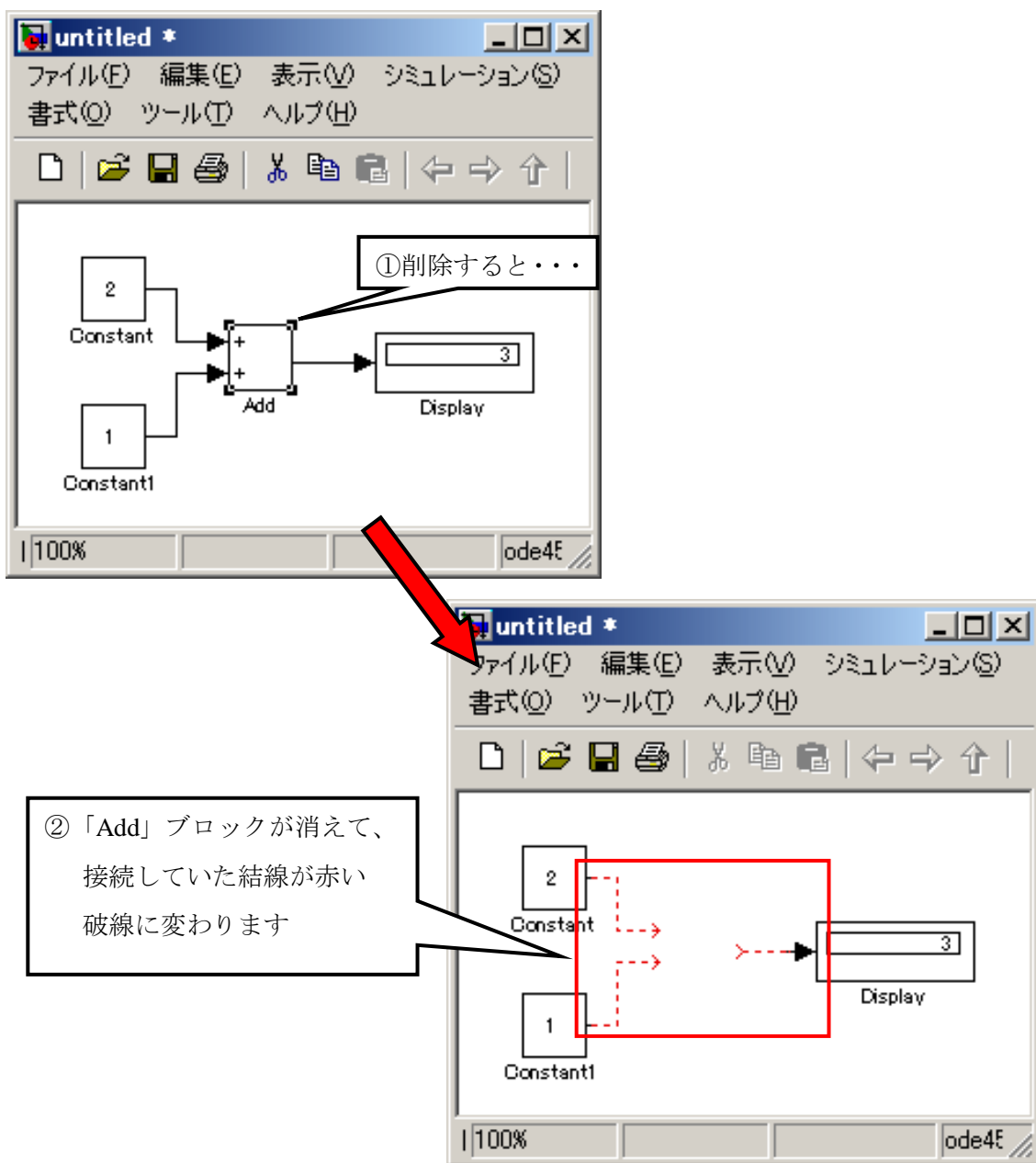


図 6-3-4 ① : ブロックの削除

先程削除した「Add」ブロックがあった箇所に「Simulink Library Browser」の「Math Operations」カテゴリにある「Subtract」ブロックを配置します。

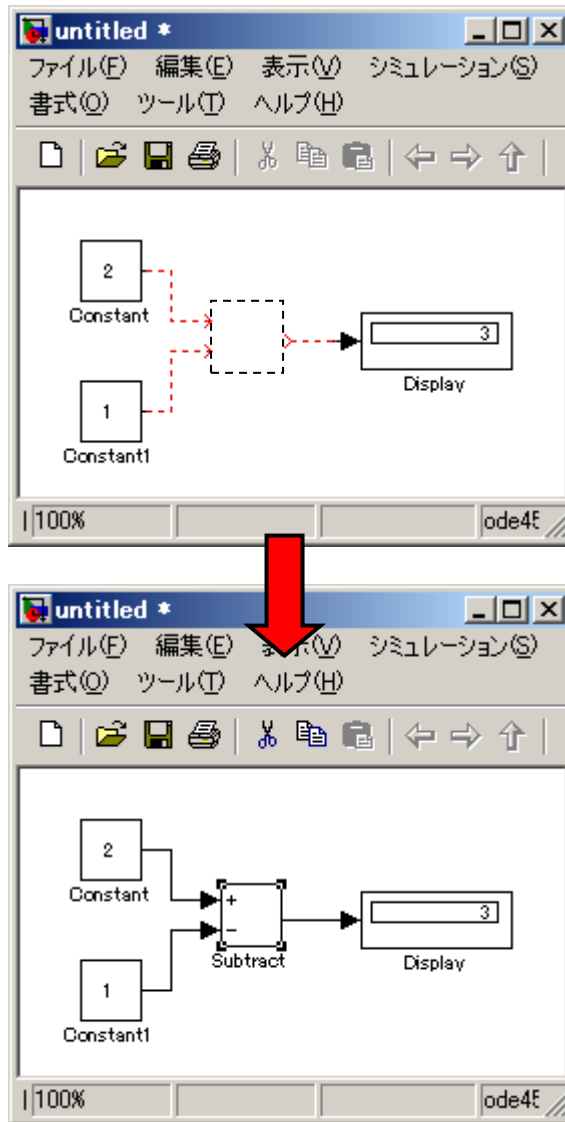


図 6-3-4 ② : ブロックの変更

配置した「Subtract」ブロックを赤い破線になっている結線の先端に接続されるようにドラッグ&ドロップします。

この時、誤った端子に結線されてしまったり、正確に結線されないことがありますので、その際は結線を削除したり、新たに結線し直して下さい。

ブロックの変更ができれば、「Constant」ブロックの定数パラメータをそれぞれ変更しながら「Subtract」ブロックの動作を確認しましょう。

「Subtract」ブロックを「Math Operations」の「Product」や「Divide」ブロックに置き換えて動作の確認もしてみましょう。

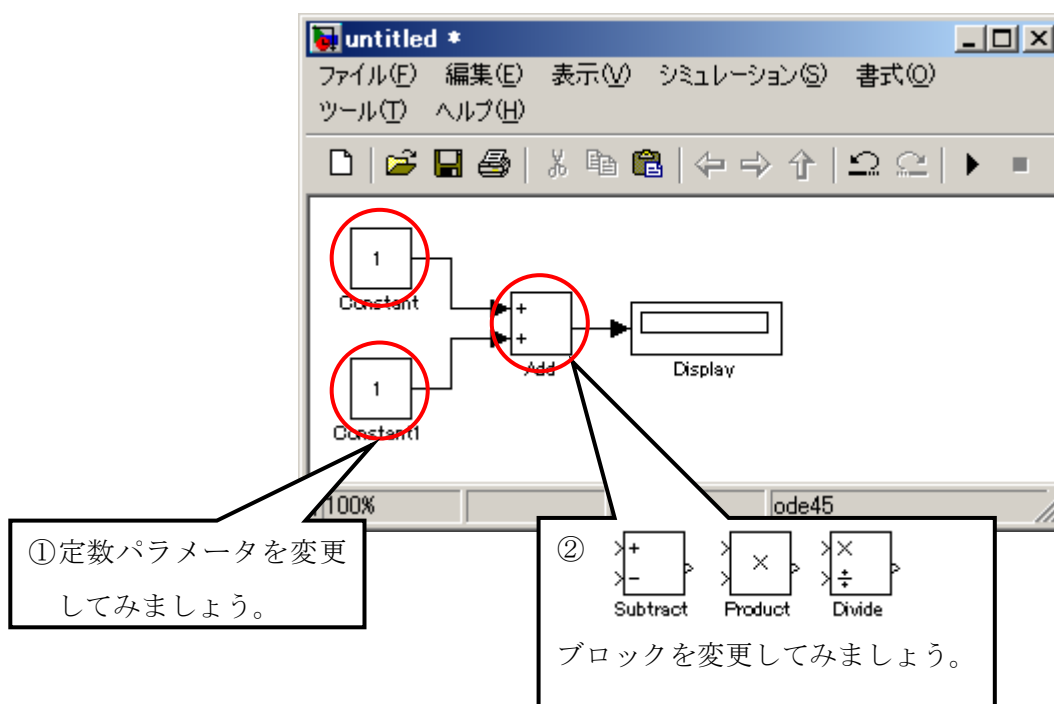


図 6-3-4 ③ : 変更後の動作確認

6. 3. 3. $(4+3) / (2+1)$ の余りを求めるモデリングをしよう。(課題1)

今までの説明を参考にして $(4+3) / (2+1)$ の余りを求める計算式を実際にモデリングしてみましょう。

ヒント：以下のブロックを利用して、モデリングして下さい。

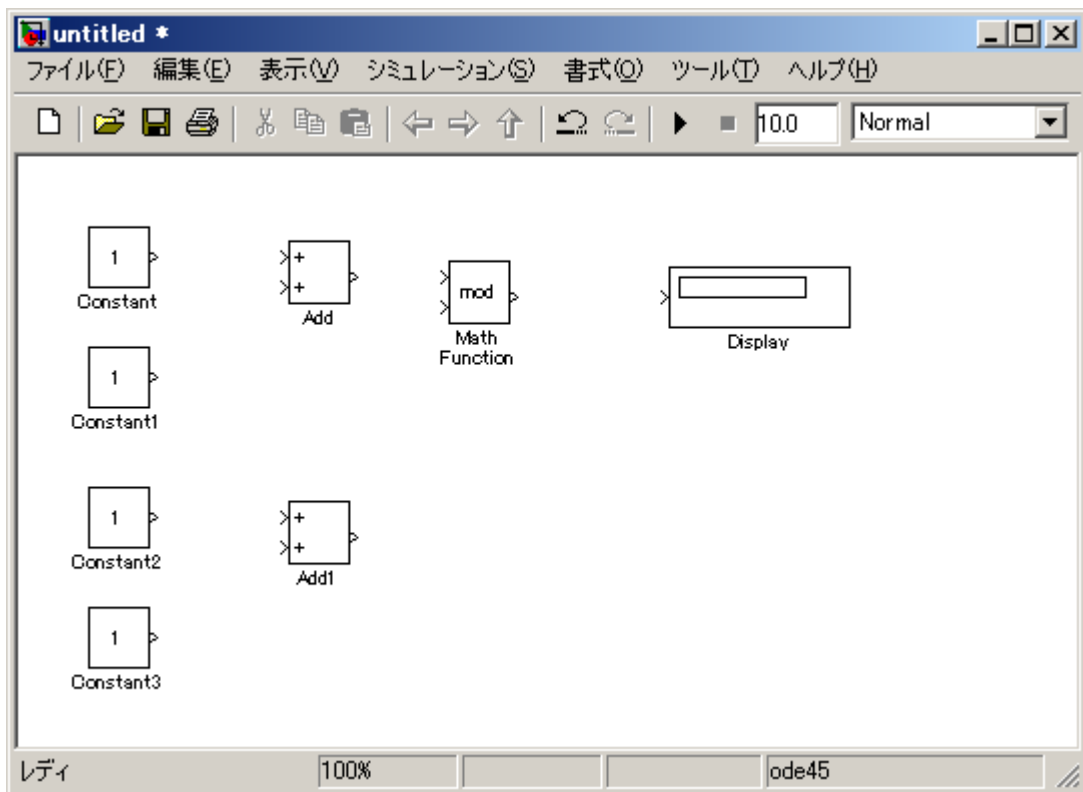


図 6-3-5 : 使用するブロック

「mod」ブロックは専用のブロックがあるわけではなく、複数の数学関数を持った「Math Function」ブロックで定義できます。

RelationalOperator や LogicalOperator 等のブロックも複数の機能を持っており、必要な関数を選択して定義できるブロックです。

6章4節 出力が変化するブロックを使ってみよう。

6. 4. 1. 出力が変化するブロックを使ってみよう。

- 出力を変化させてみよう。

「Simulink Library Browser」の「Sources」の「Pulse Generator」と「Sinks」の「Scope」を配置し、2つのブロックを結線します。

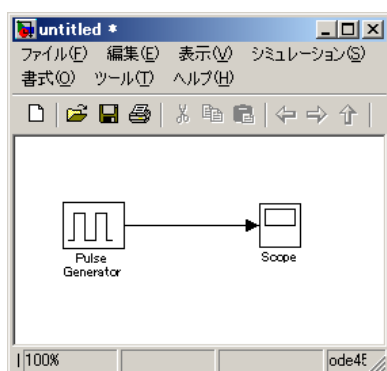


図 6-4-1 ① : ブロックの結線

シミュレーションを実行し、「Scope」をダブルクリックで開いて結果を見てみましょう。

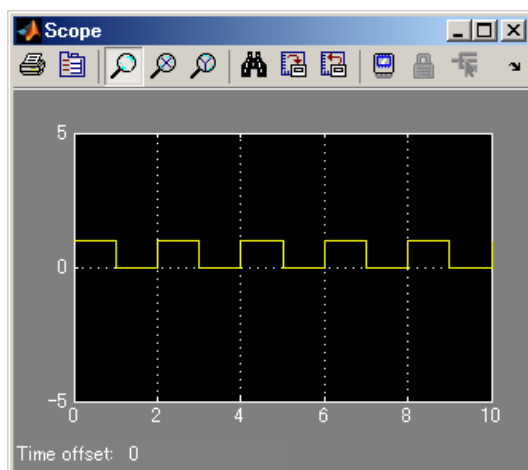


図 6-4-1 ② : 出力結果

※ 「Constant」ブロックは、時間経過に関係なく同一値を出力するブロックでしたが、「Pulse」ブロックは、時間経過に伴って出力が変化します。

「Pulse Generator」ブロックのパラメータを変更して動作を確認しましょう。

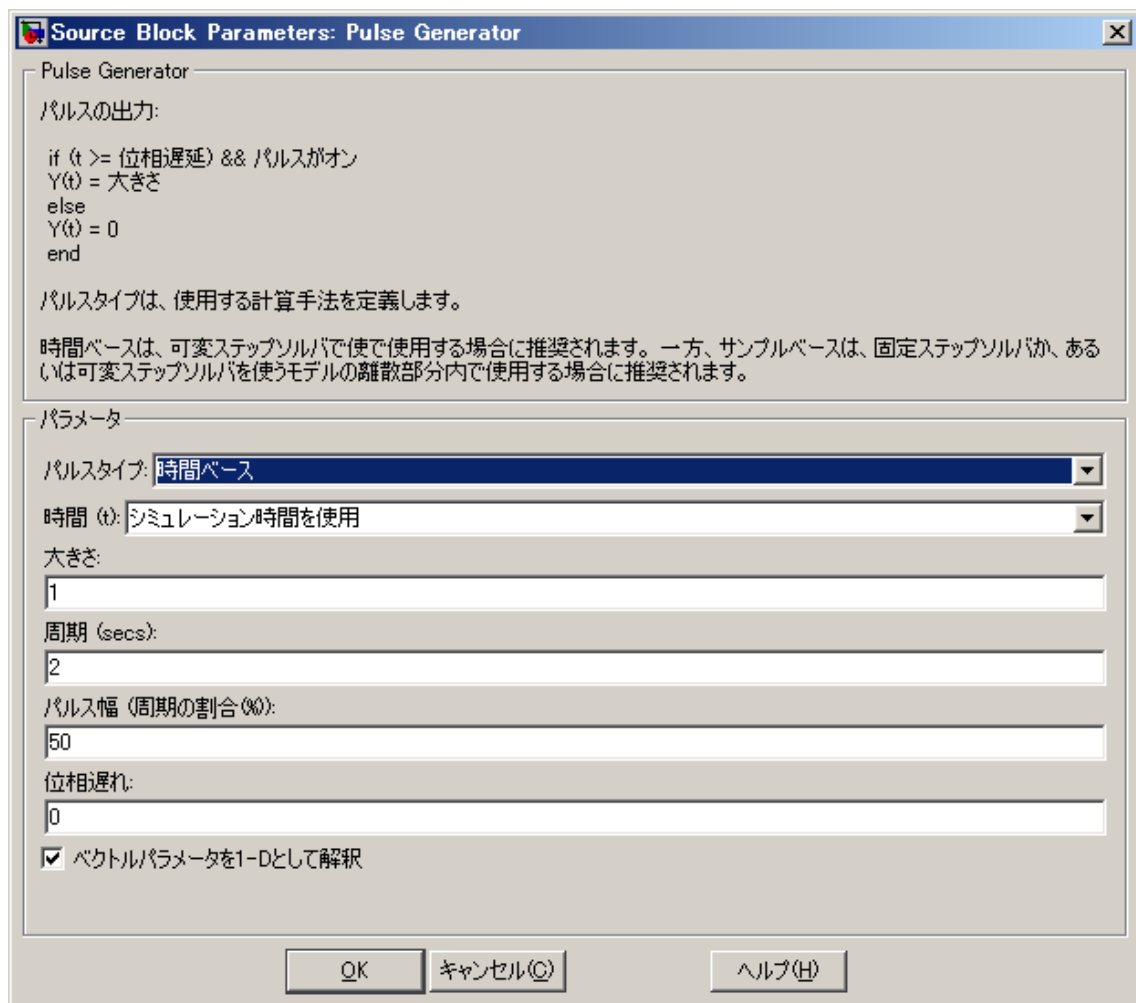


図 6-4-1 ③ : 「Pulse」ブロックのパラメータ

「大きさ」、「周期」、「パルス幅」、「位相遅れ」等が表示されるので、パラメータを変更させて、出力がどう変化するか確認してみましょう。

6. 4. 2. 出力が変化するブロックを合成してみよう。

- 複数のブロックの出力を合成してみよう。

「Pulse Generator」ブロックをコピーし、「Math Operations」の「Add」ブロックを配置し、それぞれの「Pulse Generator」の出力を「Add」の入力へつなぎます。

「Scope」ブロックを配置してパラメータを開き、「座標軸数」を「3」に変更します。



図 6-4-2 ① : 座標軸数の増加

「Scope」の入力端子が3つになったので、「Pulse Generator」、「Pulse Generator1」、「Add」の出力をそれぞれ結線しましょう。

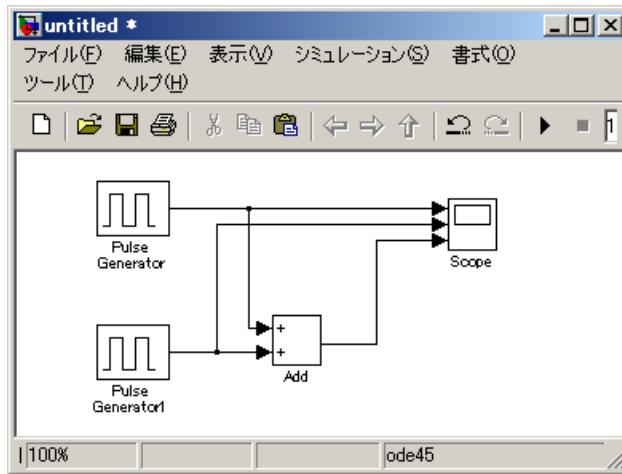


図 6-4-2 ② : 結線した図

- 合成結果を確認してみよう。

それでは「シミュレーション」を実行し、「Scope」ブロックをダブルクリックして結果を確認してみましょう。

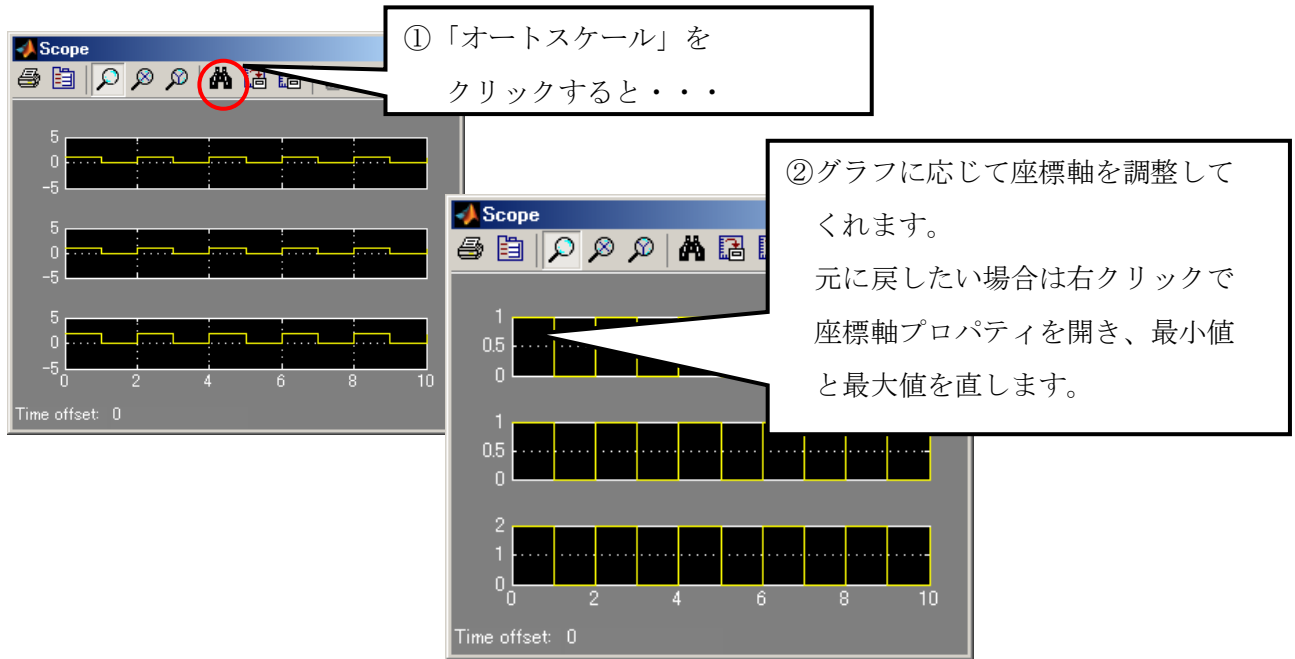


図 6-4-3 : オートスケール

上の波形は、「Pulse Generator」、真中は、「Pulse Generator1」、下は、「Add」の波形です。「Add」の波形は、振幅が「2」になっており加算されたことが確認できます。

- 変更後の出力を確認しよう。

「Pulse Generator1」のパラメータを変更して合成出力の変化を確認してみましょう。

「Pulse Generator1」のパラメータを以下のように順次、変更し、合成出力を確認しましょう。

1. 「大きさ」を「1」から「2」へ変更し、確認。
2. 「周期」を「2」から「4」へ変更し、確認。
3. 「位相遅れ」を「0」から「1」へ変更し、確認。
4. 「パルス幅 (%)」を「50」から「25」へ変更し、確認。

6章5節 いろいろなブロックを使ってみよう

・カウンターモデルを作成してみよう。

「Discrete」カテゴリ内の「Unit Delay」ブロックは入力されたデータを1サンプル分遅らせて保持し、次のサンプリングで出力します。ブロックに対する入力がベクトルの場合、ベクトルのすべての要素は同じサンプル遅れ分だけ遅らせます。

以下のモデルは「Unit Delay」ブロックを利用したカウンターです。1サンプル遅れを利用してカウントを1ずつ増やしています。

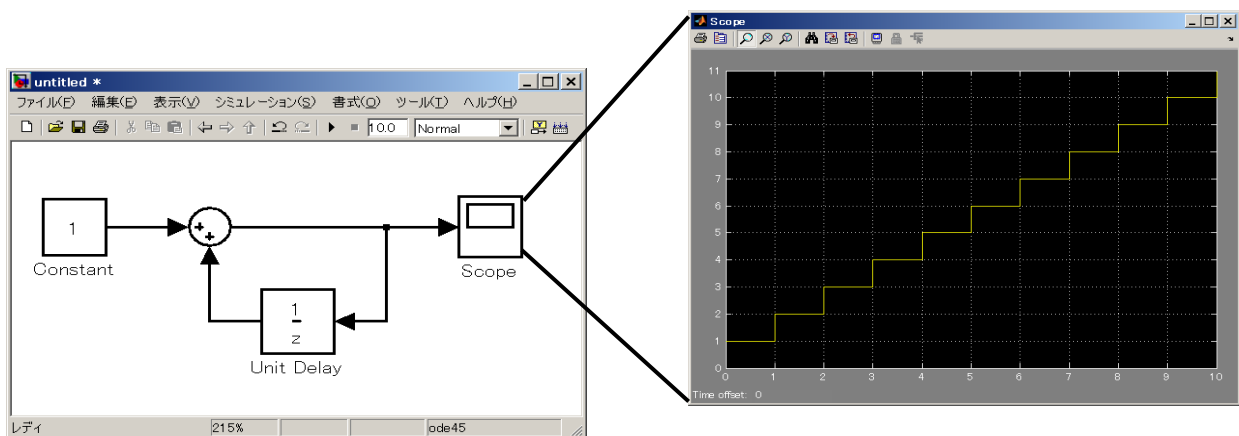


図 6-5-1 ① : カウンターモデル

また、ここでは右から信号が入力され、左へ出力されています。基本的には信号は左から右ですが、フィードバックのような出力を入力に戻す時等に端子を反転させて、右から入力させ、左から出力ができます。

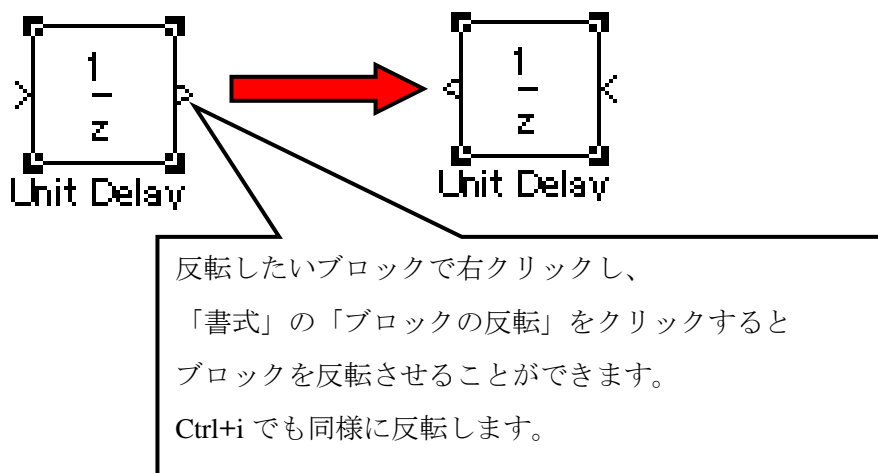


図 6-5-1 ② : ブロックの反転方法

・ 正弦波を出力してみよう。

「Sources」カテゴリ内の「Sine Wave」ブロックを使用することで正弦波を出力することができます。

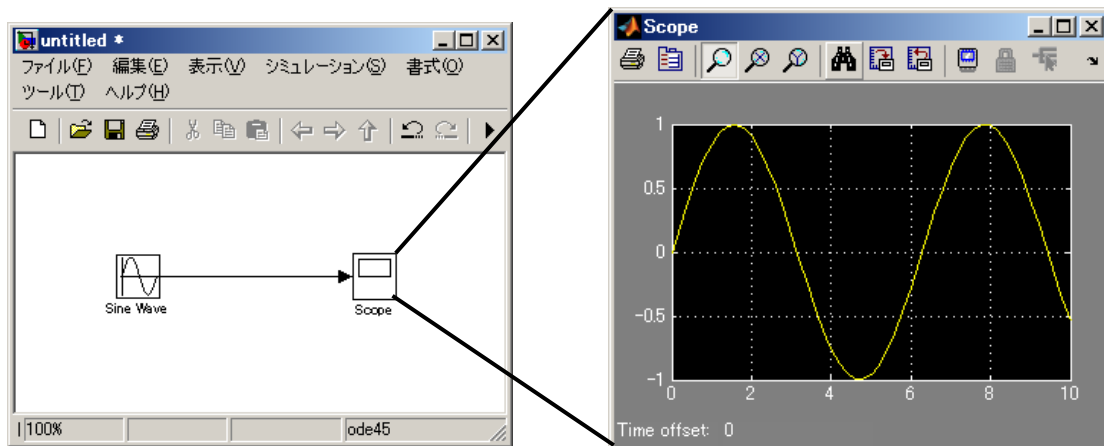


図 6-5-2 ① : 正弦波出力モデル

「Sine Wave」ブロックのパラメータで正弦波の振幅や周波数等の設定を行うことができます。

パラメータ設定は今回に限らず数式で行うことができます。例えば周波数なら 24 時間をラジアン変換して図のように記述できます。

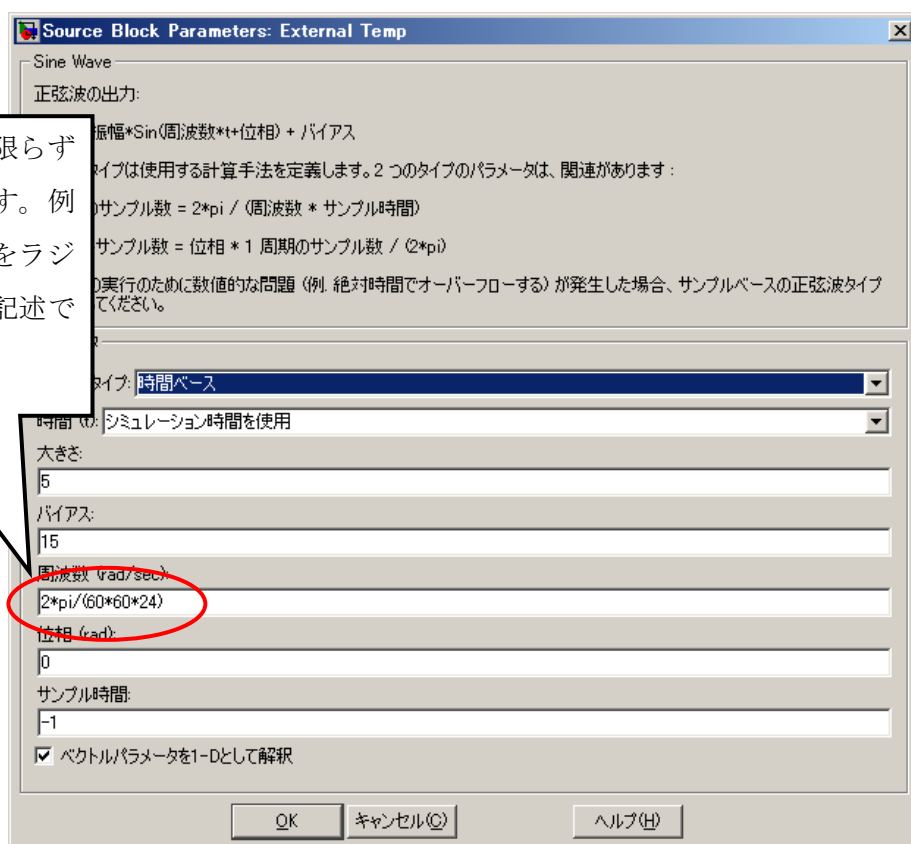


図 6-5-2 ② : パラメータの設定

• 信号を多重化してみよう。

複数の信号線を1つに多重化する場合に、「Mux」ブロックを使用することで信号を多重化することができます。「Mux」ブロックは「Signal Routing」カテゴリ内にあります。

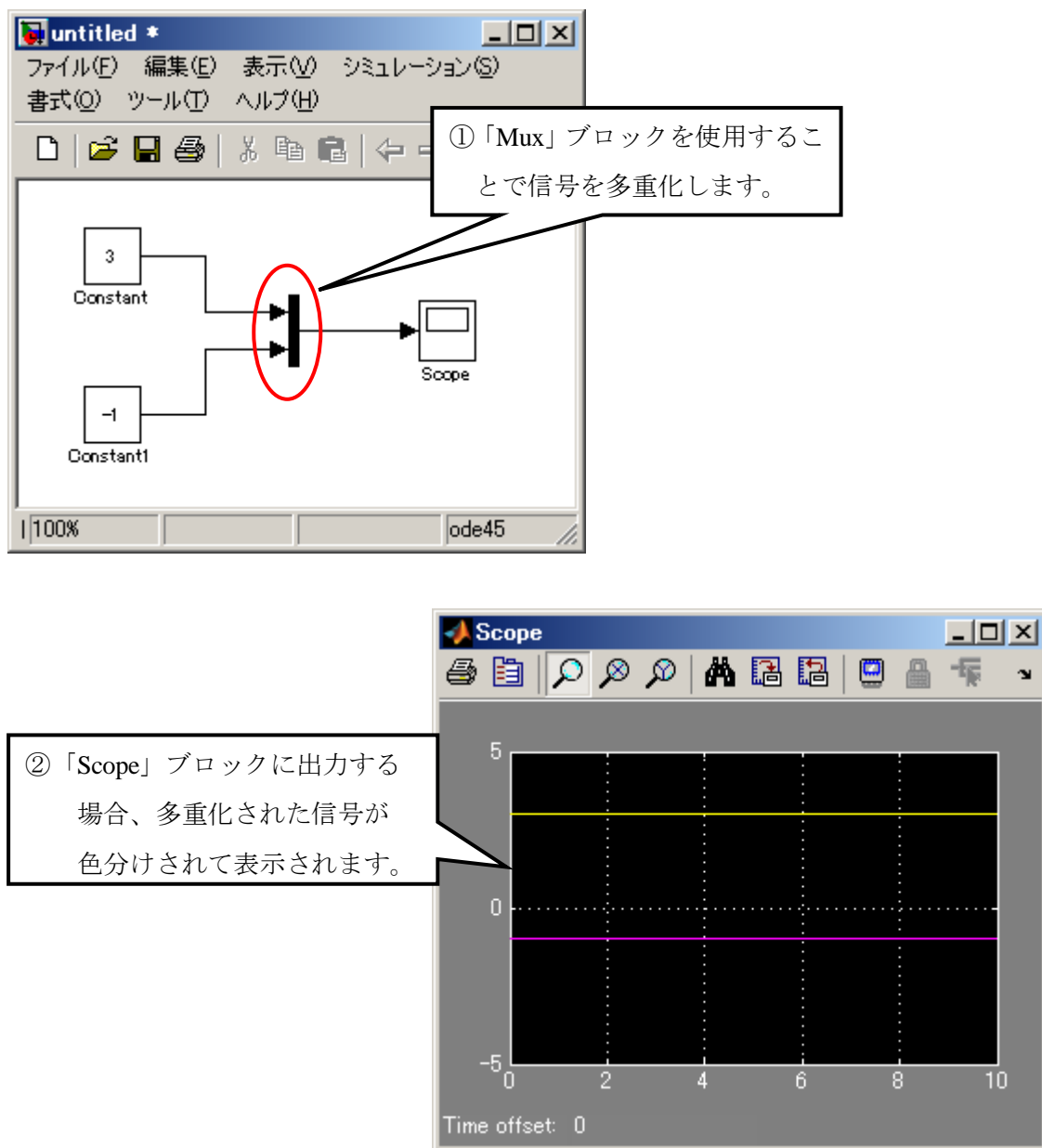


図 6-5-3 : 信号の多重化モデル

• 信号名を入力しよう。

ブロック間の結線を行った際に、信号線に信号名を入力することが推奨されています。信号名を加えることでより可視性が向上されます。

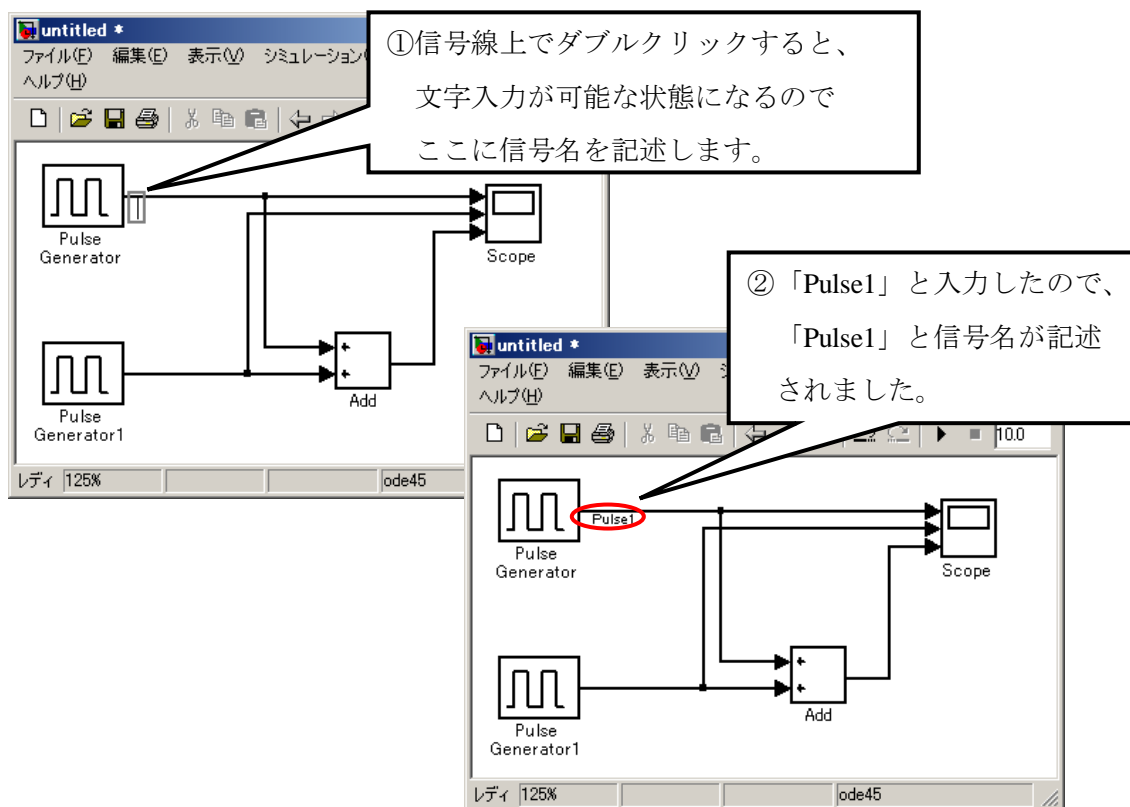


図 6-5-4 ①：信号名の入力

また、信号名を入力すると「Scope」で出力する際に、信号名の文字列が継承されて表示されます。

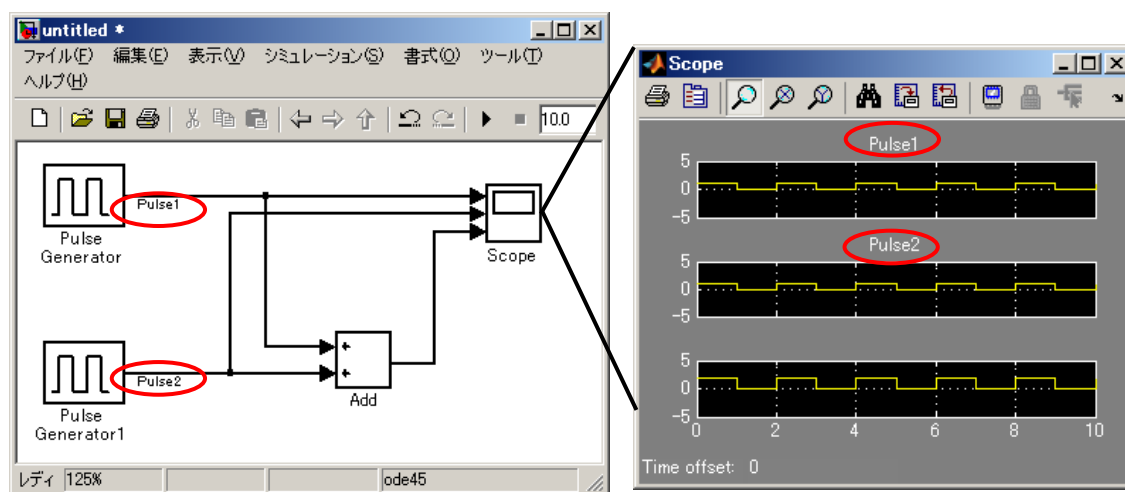


図 6-5-4 ②：Scope での信号名の確認

・グラフ名を記述しよう。

信号名が記述されている場合はそちらが継承されてグラフ名として表示されますが、記述されていない場合にグラフ名を直接記述することができます。

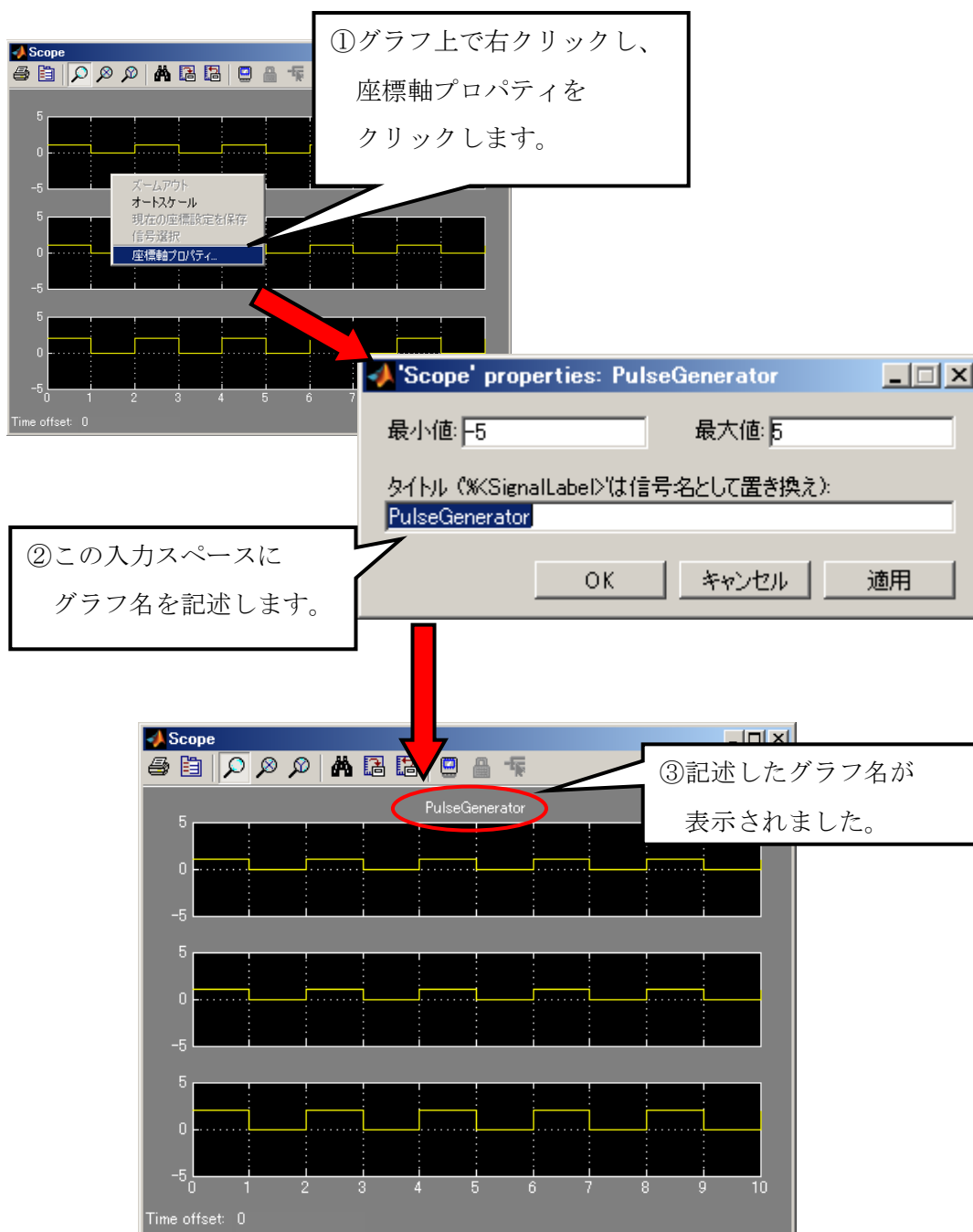


図 6-5-5 : グラフ名の記述

・「Scope」ブロックのパラメータを変更するには？

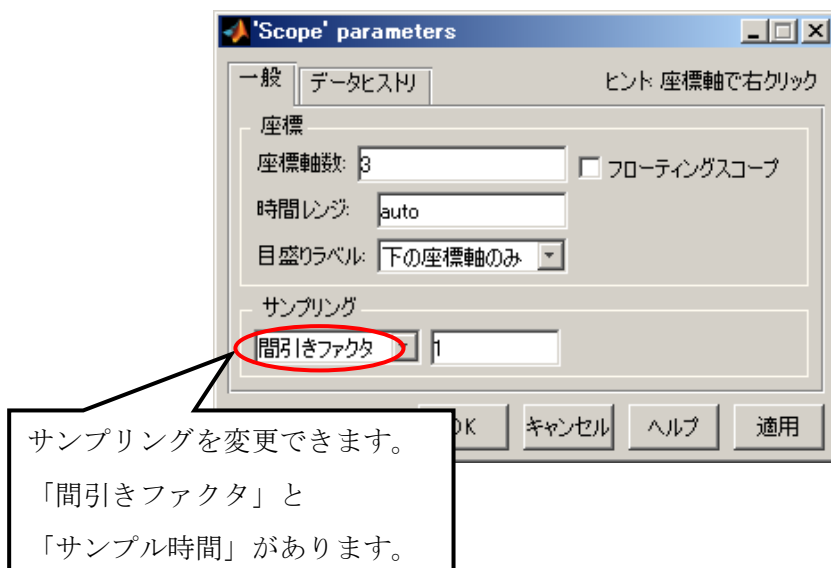


図 6-5-6 ①：間引きファクタの変更

「間引きファクタ」のテキストフィールドに数値を入力することで出力データの間引き率（倍数指定）を行います。デフォルトは1になっていて間引きを行いません。

「サンプル時間」を選択し、数値を入力することでサンプリングを行う時間の間隔を設定します。デフォルトでは0になっています。

「Scope」ブロックにはデータ点の制限があり、指定されたデータ点数分の最新データ分だけが表示されます。つまり、制限を超えるシミュレーションを行った場合は、全て表示されずにシミュレーションの終わりの方のデータしか表示されません。

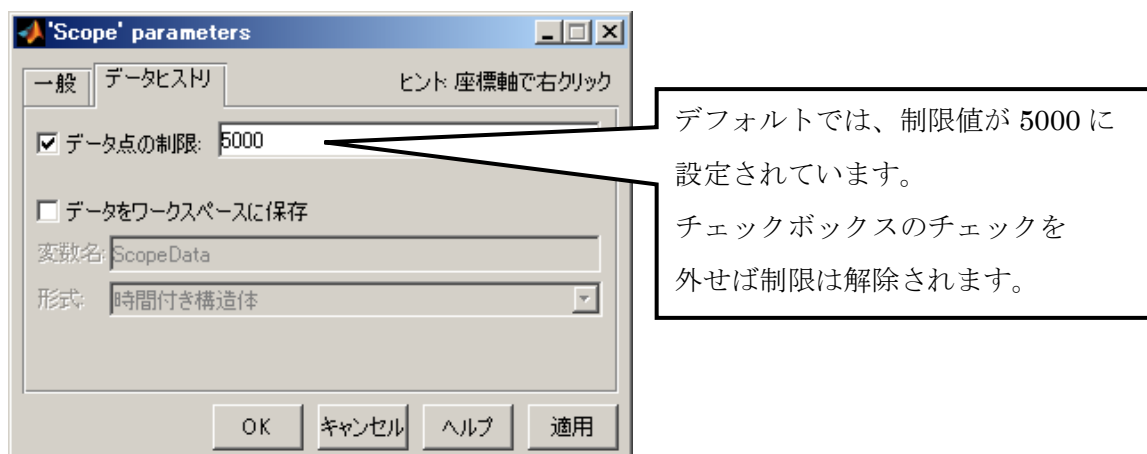


図 6-5-6 ②：データ点の制限の変更

• 任意の波形を作成してみよう。

「Signal Builder」ブロックは、波形を作成できるブロックです。

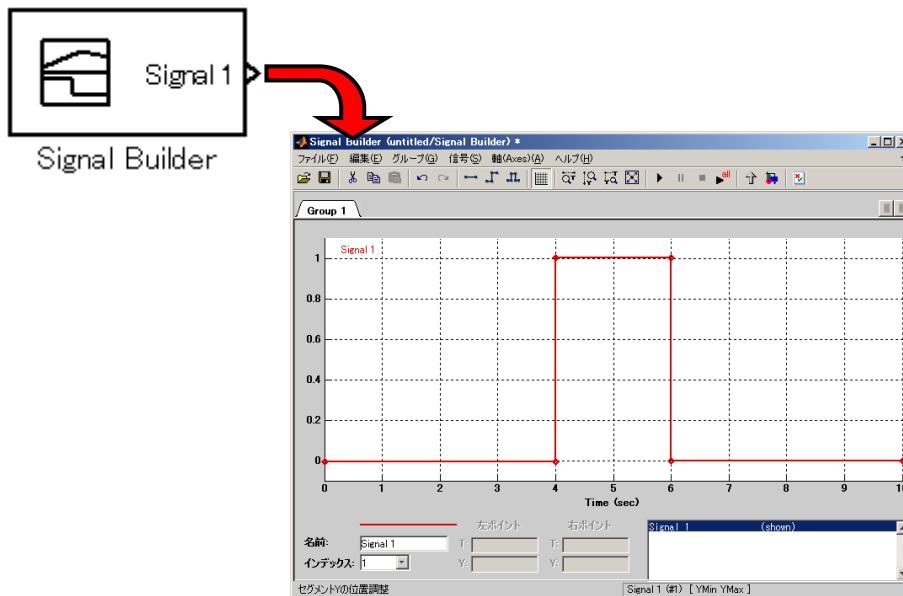


図 6-5-7 ① : Signal Builder の画面

「Signal Builder」ブロックを配置してダブルクリックすると、上図のような画面が開きます。ここで波形を作成することができます。

④信号を追加したい時は対応したアイコンをクリックすれば定数信号、ステップ信号、パルス信号が追加できます。

②ラインはY方向のみドラッグで変更が可能です。

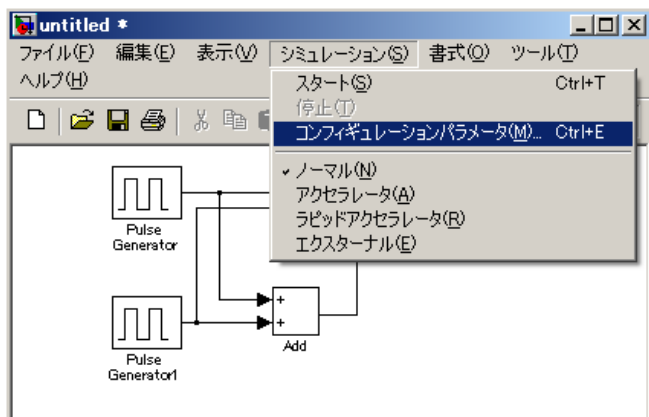
①新しくポイントを作成する時は、ライン上でShift+クリックします。削除する時は選択してDeleteを押します。

③座標を微調整する時はここに数値を入力します。

図 6-5-7 ② : 任意の波形の作成

・コンフィギュレーションパラメータを設定しよう。

コンフィギュレーションパラメータでは、シミュレーション時間やソルバ設定を行います。コンフィギュレーションパラメータを設定するには、モデルウィンドウの「シミュレーション」メニューから「コンフィギュレーションパラメータ」を選択します。



ここで任意のシミュレーション時間の設定をします。
終了時間に「inf」と設定すると停止ボタンの■を押すか、
「シミュレーション」の「停止」を押すまで
シミュレーションし続けます。

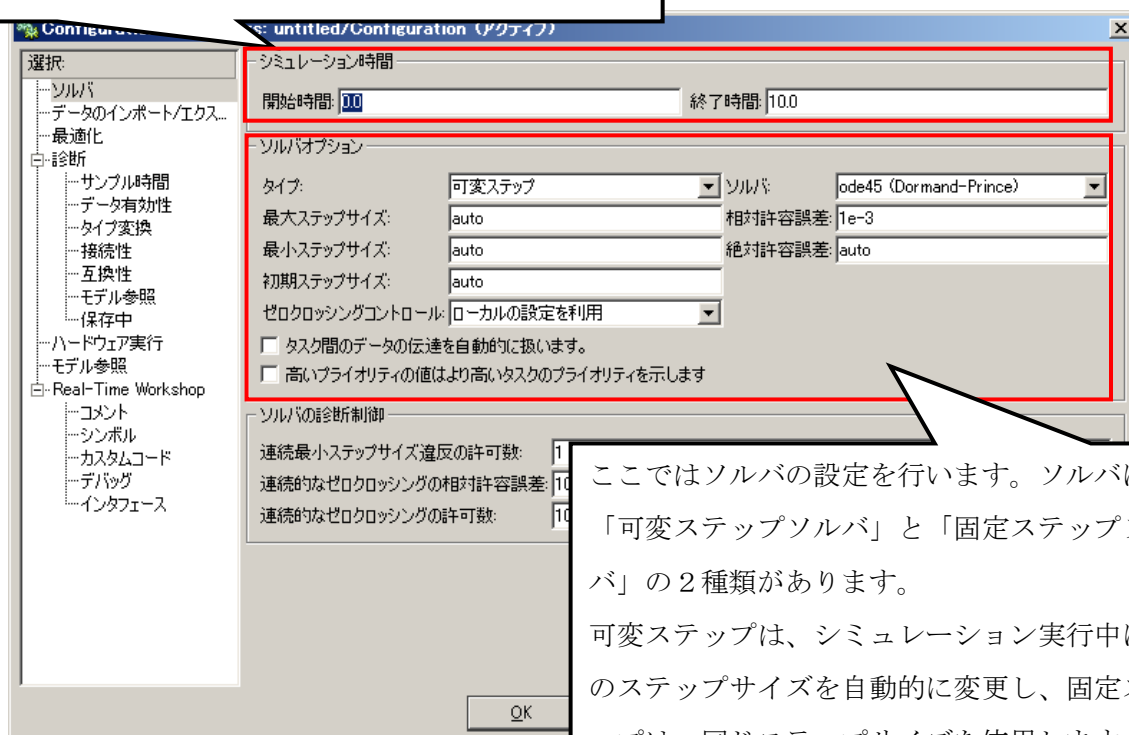


図 6-5-8 : コンフィグレーションパラメータの設定

7章 実際の開発とモデリングとの関係



7章 1節 機能と制御システムモデルの関係

機能を作ります ⇒ 機能は、電子制御によって実現されます
⇒ 機能を統合することにより、新しい機能を創造します（機能統合）

複雑化、高速化する自動車制御システム開発は、モデリングによる仮想化された開発環境によって、品質向上、開発効率向上が可能となります。

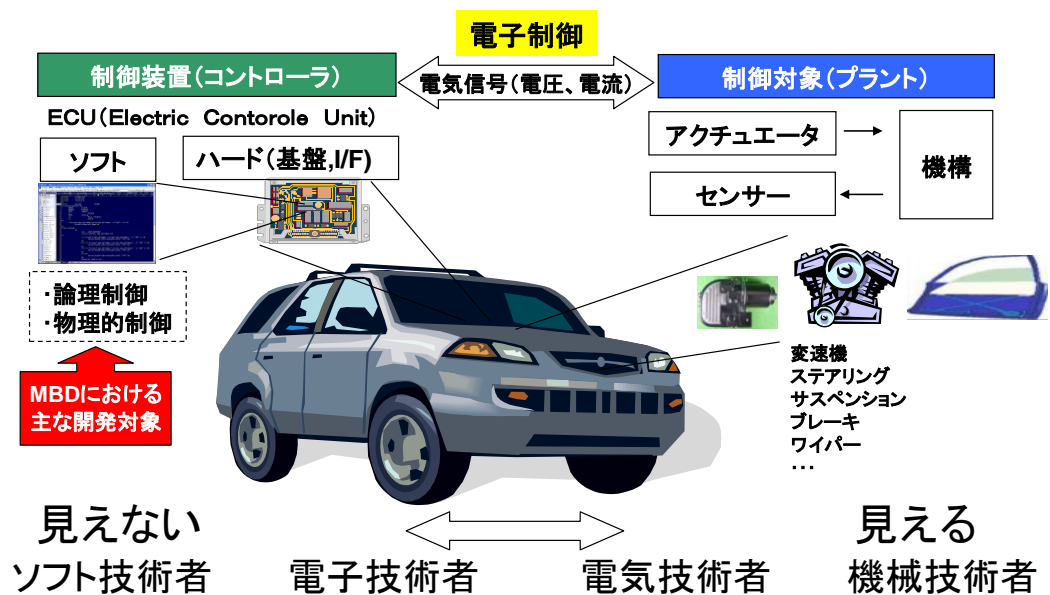
MBD では、機能設計、検証、情報共有をモデルにより統合的に扱えます。

構成要素の階層	機能の特性
車両	人間寄り、統合的、論理的、抽象的、マクロ ⇕ 機械寄り、独立的、現実的、具体的、ミクロ
システム	
ユニット	
パーツ	

階層ごとに求められる機能（仕様表現）が異なります。 ⇒ 階層間の情報共有。

MBD は、異なる機能の特性を持つ構成要素を一連のモデルとして統合的に扱えます。

7章2節 実際の制御システムの構成（自動車）



MBDでは、上図のように、制御をするものを「制御装置（コントローラ）」、制御されるものを「制御対象（プラント）」と定義しています。

7章3節 制御システム（制御装置）における二種類の制御

制御システムまたは制御系（英：Control system）とは、他の機器やシステムを管理し制御する為の機器、あるいは機器群である。制御システムは大まかに、論理制御（逐次制御）とフィードバック制御（線形制御）に分類される。

(Wikipedia 2008.12.1)

1. 論理制御（逐次制御）

「システムのふるまい」はロジックフローとして表現され、予め決められたシーケンス（ロジック）に従って逐次、実行される。

MBDでは、ロジックフローはシステムを有限状態機械（有限オートマトン）として捉え、有限個の状態と遷移と動作の組み合わせとして表現する状態遷移図、状態遷移表で主にモデリングする。

※システムとは、相互に影響を及ぼしあう要素から構成される、まとまりや仕組みの全体

2. フィードバック制御（線形制御） ⇒ 動的制御

フィードバック制御は要求された目標値、制御対象からのフィードバック値、その差分データ等の変化によって駆動されるデータフロー制御です。

MBDではこれら2種類の制御を主なモデリング対象とする。

7章4節 制御装置（コントローラ）

■ 制御装置（制御システム）

他の機器（**制御対象**）の**制御**や**システム全体を管理**する為の機器である。制御システムは大まかに、**論理制御**（逐次制御）と**フィードバック制御**（線型制御）に分類される。
(Wikipedia 2008.10.28)

動的制御（フィードバック制御等）

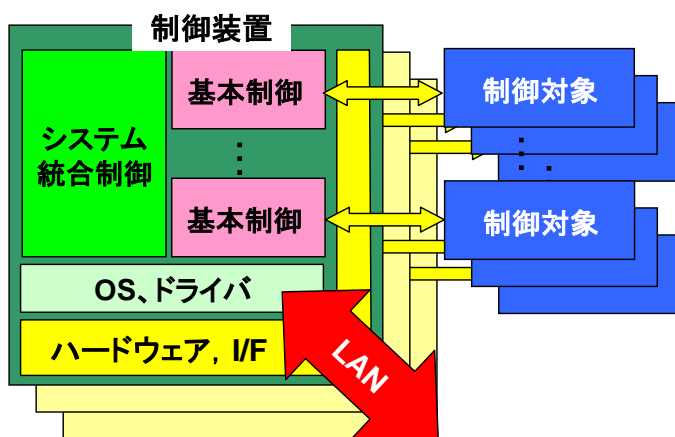
制御理論や**実験、検証**によって**最適化された制御アルゴリズム**によって、**直接、制御対象を制御**する部分である。

データフローで実現する。

論理制御

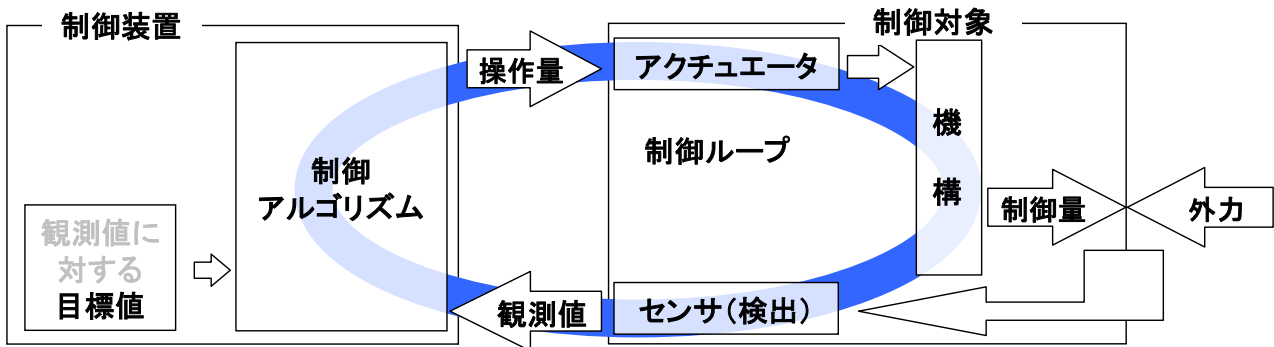
システムのふるまいを、状態遷移として制御する論理制御部である。
動作として動的制御部へ制御指令が発せられる。

ロジックフローで実現する。



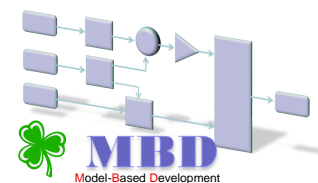
7章5節 フィードバック制御

(線型)フィードバックシステムには、制御アルゴリズムとアクチュエータとセンサから成る「**制御ループ**」があり、**何らかの変数が標準値(目標値)になるよう制御**する。(Wikipedia)



フィードバック制御: 観測値が目標値と一致するように操作量を操作する制御である。

8章 ON/OFF 制御モデルのモデリング

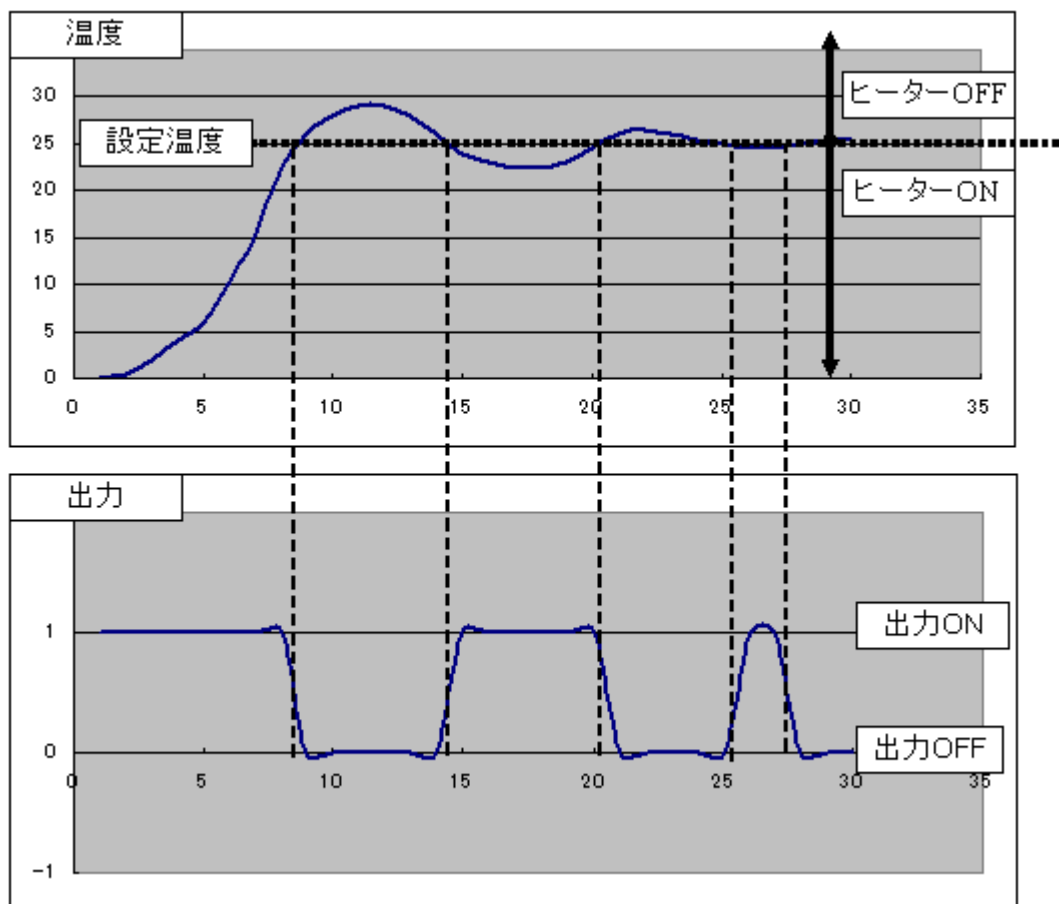


8章1節 ON/OFF 制御

ON/OFF 制御は、目標値と観測値（制御値）を比較し、その結果から操作値を出力する（ON）か、出力しないか（OFF）切り替える制御方式です。

例) ON/OFF 制御で温度を 25°C（設定温度）に制御する場合、

温度が 25°C より低い場合には、ヒーター出力を ON し、
温度が 25°C より高い場合には、ヒーター出力を OFF することで、
温度を一定にしています。

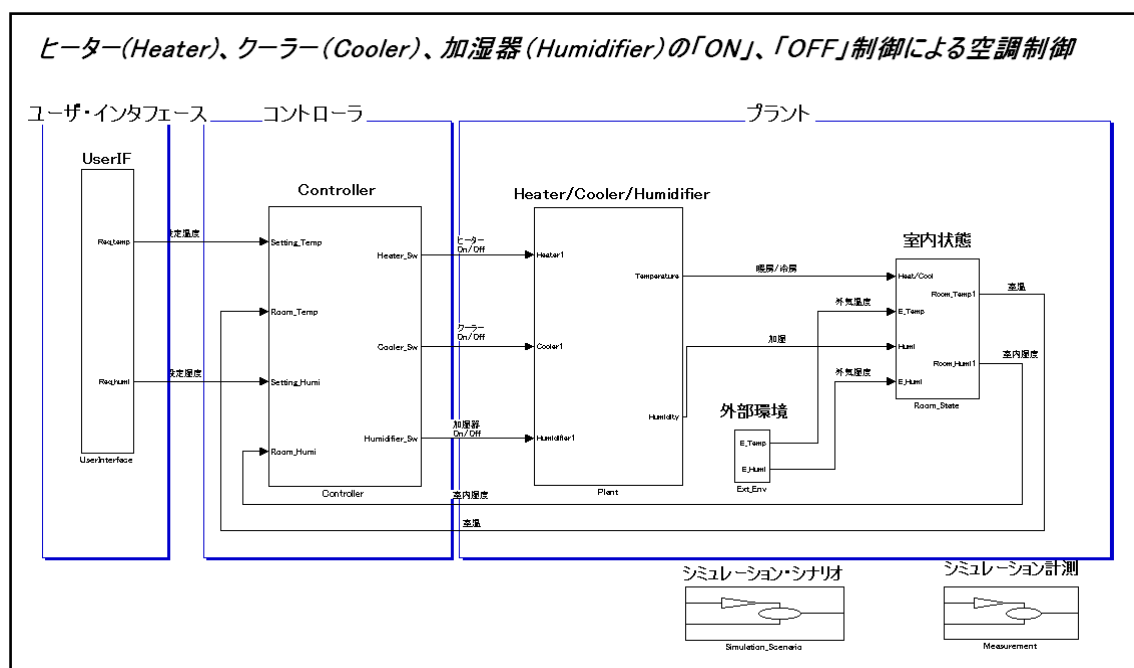


8章2節 ON/OFF 制御モデルのモデリング

8. 2. 1. 簡易オートエアコンのサンプルモデルを開いてみよう。

簡易オートエアコンのサンプルモデルが入っているファイルを開いて見ましょう。

(添付の CD-R の [8 章 ONOFF 制御モデルのモデリング¥AutoAirControl¥ベースモデル] フォルダ内から [air_control.mdl] をデスクトップ上にコピーし、ファイルを開いて見ましょう。)



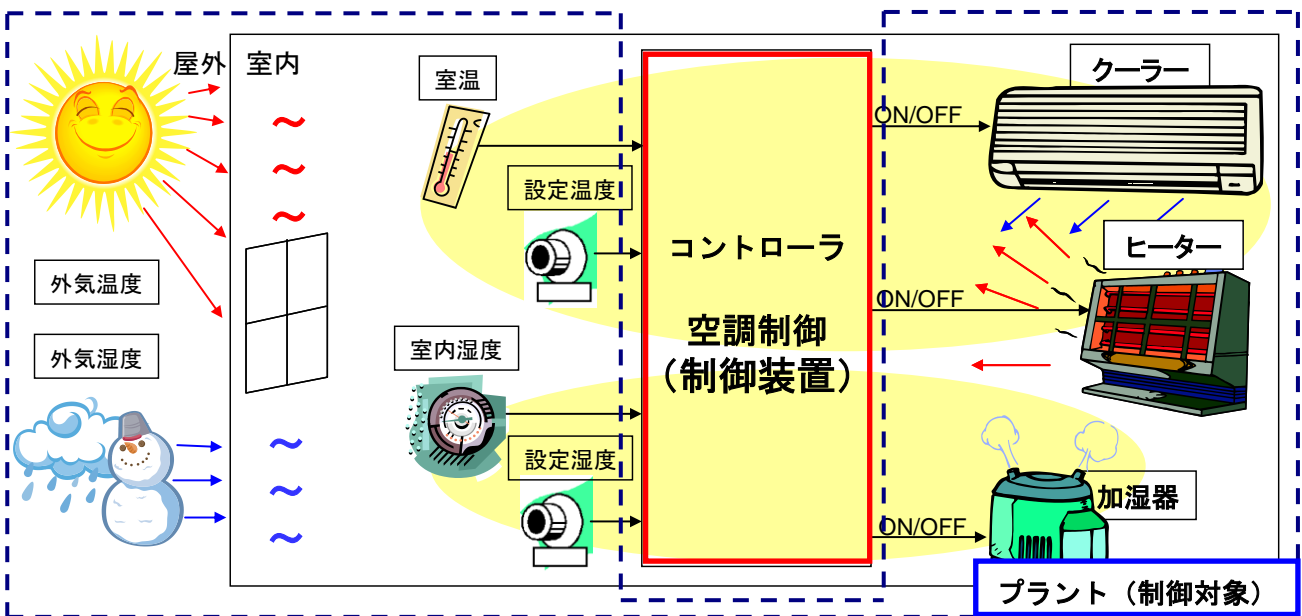
8. 2. 3. 簡易オートエアコンの制御仕様

クーラー、ヒーター、加湿器の「ON/OFF 制御」

制御仕様

1. 室温が設定温度より高い場合は、クーラーを「ON」にし、ヒーターを「OFF」にする。
2. 室温が設定温度より低い場合は、クーラーを「OFF」にし、ヒーターを「ON」にする。
3. 室内湿度が設定湿度より高い場合は、加湿器を「OFF」にする。
4. 室内湿度が設定湿度より低い場合は、加湿器を「ON」にする。

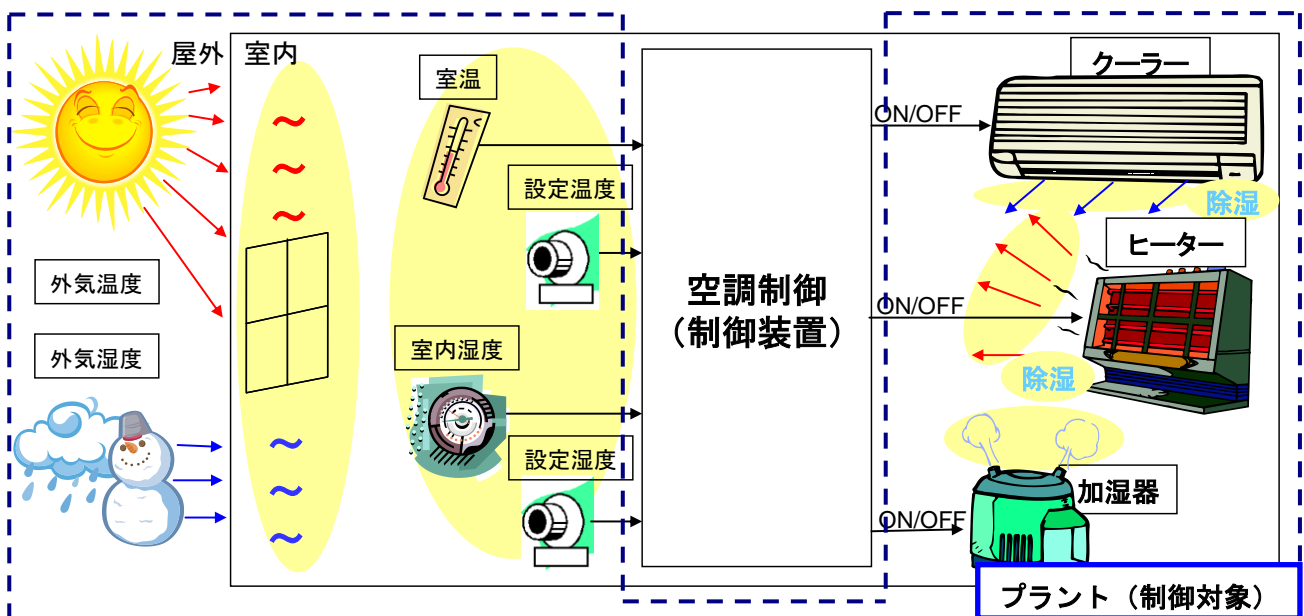
コントローラとプラント (コントローラ以外)



8. 2. 4. プラントの解説

・下記要素に対して簡易モデリングしてあります。

1. クーラー、ヒーター、加湿器の冷房、暖房、加湿能力
2. クーラー、ヒーター稼働による間接的な除湿効果の影響
3. 外気の温度変化、湿度変化による室温、室内湿度への影響



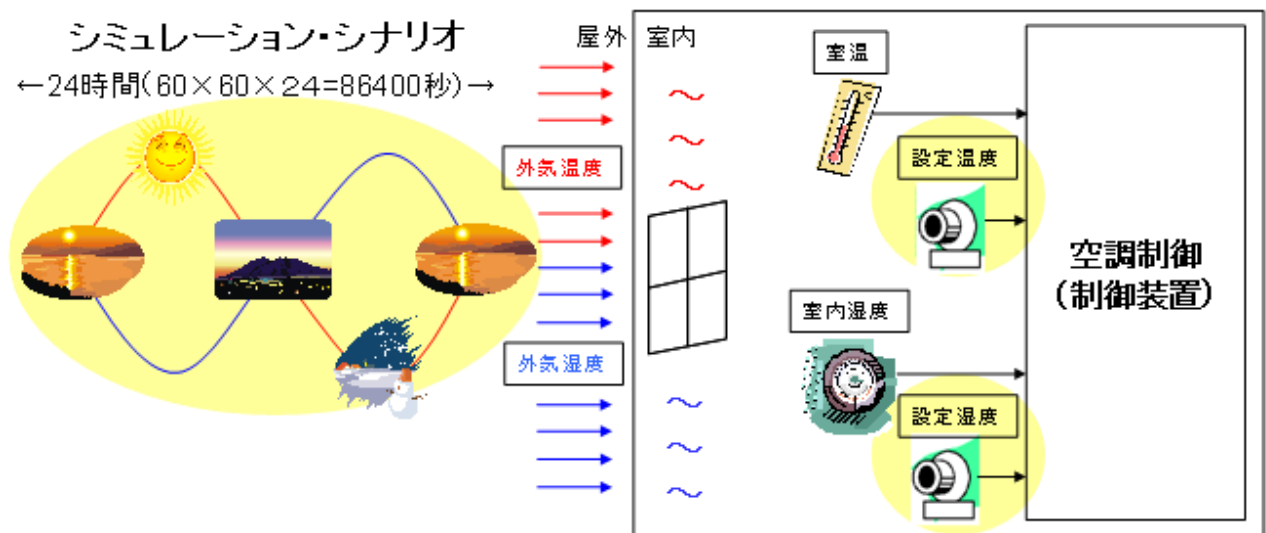
8. 2. 5. 簡易オートエアコンのシミュレーション（シナリオ）解説

・下記要因に対して簡易的にシミュレーションができるよう設定されています。

1. 外気温度と外気湿度を24時間の実測値として表現しています。

温度と湿度は、群馬県前橋市の1日の観測結果が設定されています。

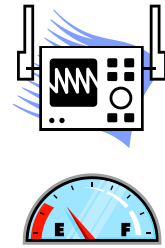
2. 設定温度、設定湿度が設定できます。



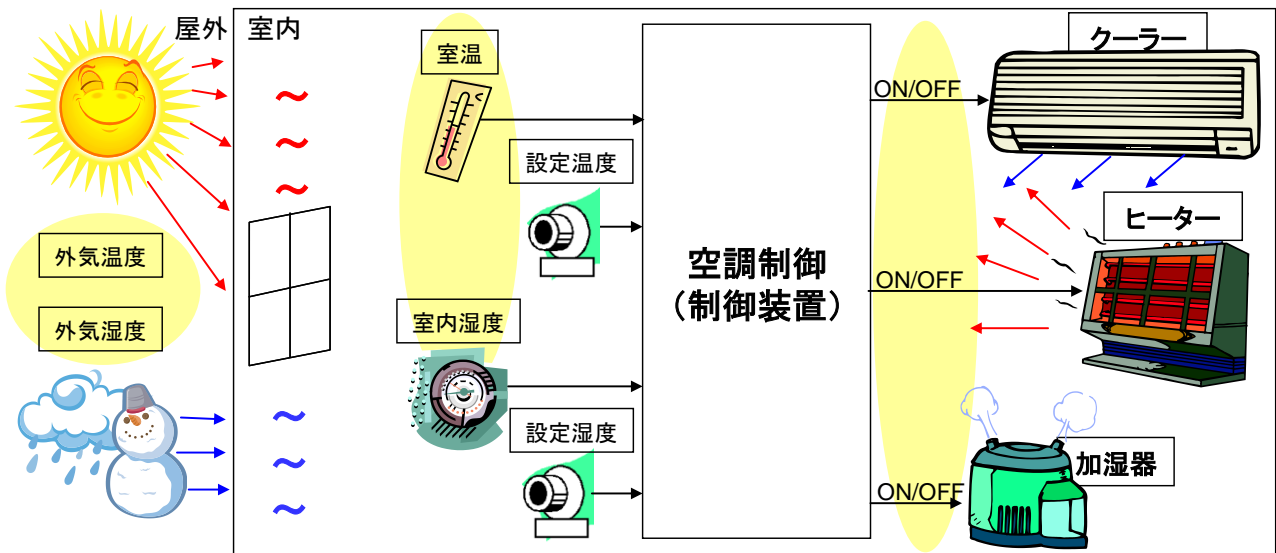
8. 2. 6. 簡易オートエアコンのシミュレーション（観測）解説

・下記要素に対して観測できるようにしてあります。

1. 外気温と室温、ヒーター、クーラーの ON/OFF 状態、
外気湿度と室内湿度、加湿器の ON/OFF 状態を 24 時間分、
観測できます。



2. ヒーター、クーラー、加湿器のそれぞれの稼働時間から 24 時間の消費電力の
合計を観測できます。



8章3節 ON/OFF 制御を実際にモデリングしてみよう。(課題2)

- ・コントローラサブシステムを開いて、ON/OFF 制御のコントローラを完成させましょう。

8章4節 簡易オートエアコンの省エネ化（課題3）

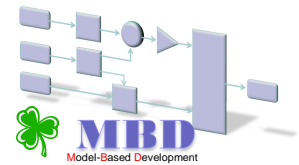
課題：簡易オートエアコンの省エネ化に挑戦してみよう。
(注意：プラントモデル部分を変更しないでください。)

ヒント：現状、ヒーターかクーラーのどちらかが必ず動いています。
設定温度以下に冷えるとヒーターが、設定温度以上に温まると
クーラーが動き始めます。
設定値への復帰には「自然の力」（外気温）を利用しましょう。
(コントローラモデルには ExtTmp が入力してあります。)

検証1：シミュレーションモデルの消費電力量をオリジナルのモデルと
比較しよう。
検証2：シミュレーションモデルの室温、室内湿度の特性の変化を
オリジナルのモデルと比較しよう。

別の方法で省エネを考え出してみよう。

9章 PID 制御モデルのモデリング



9章1節 PID 制御

9. 1. 1. PID 制御の基本式

PID 制御基本式は、下記の式で表されます。 ← 定義、理論

$$y = K_p (e + 1/T_i \cdot \int e dt + T_d \cdot de/dt)$$
$$= K_p \cdot e + K_p/T_i \cdot \int e dt + K_p \cdot T_d \cdot de/dt$$

比例動作 積分動作 微分動作

y : 操作量
e : 偏差 (=目標値-観測値)
K_p : 比例ゲイン
T_i : 積分時間
T_d : 微分時間

●比例動作 (Proportional Action : P 動作)

現在の偏差 e に比例した操作量を出力します。

偏差が大きければ、操作量を大きく、小さくなれば、操作量を小さくします。

⇒ 目標値に段階的に近づける操作量を出力します。

しかし、最後には残留偏差が残ってしまいます。

●積分動作 (Integral Action : I 動作)

P 動作の残留偏差の除去を行います。

過去の偏差の累積値に比例した操作量を出力します。

累積値が大きくなれば、操作量を大きくします。

⇒ 目標値に一致させる操作量を出力します。

●微分動作 (Derivative Action : D 動作)

P、I 動作は、偏差の急激な変化に対して応答速度が遅くなります。

微分動作では、偏差 e の変化率に比例した操作量を出力します。

⇒ 偏差の急激な変化にも即座に追従する操作量を出力します。

PID 制御は、これら 3つの動作を加算合成したもので表現されます。

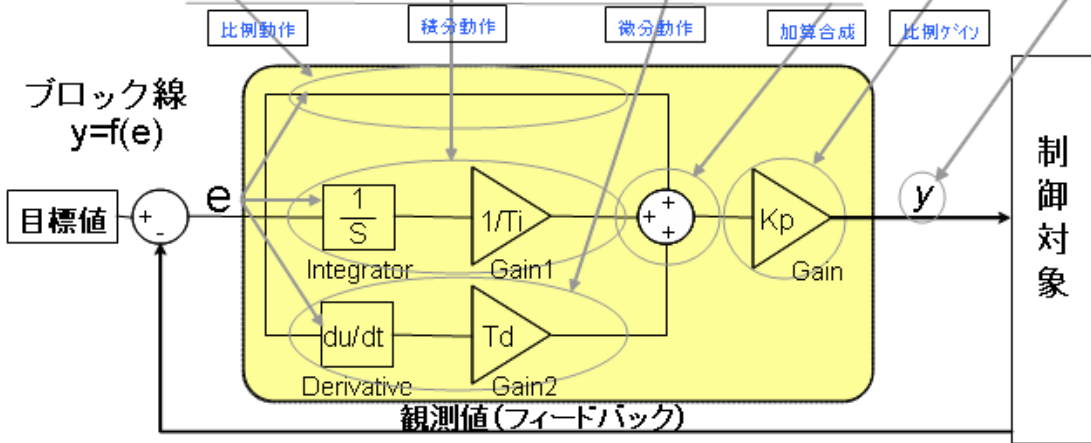
9. 1. 2. PID 制御の基本式のモデリング

PID制御式は下図のような対応付けで、
ブロック線図(モデル)として表現できる。

y : 操作量
 e : 偏差 (= 目標値 - 観測値)
 K_p : 比例ゲイン
 T_i : 積分時間
 T_d : 微分時間

$$y = K_p(e + 1/T_i \cdot \int e dt + T_d \cdot de/dt) \dots\dots ①$$

$$(e + \int e dt \cdot 1/T_i + de/dt \cdot T_d) \cdot K_p = y \dots\dots ②$$



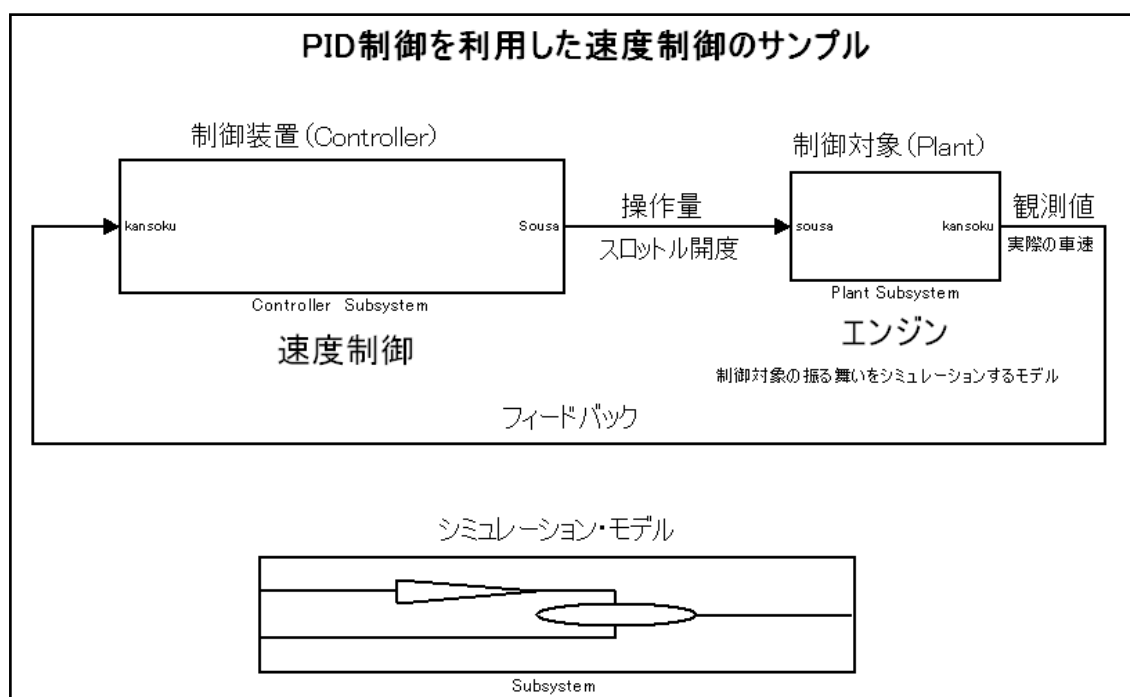
モデルの定義: 対象の機能が図示されており、一意的に解釈ができるもの

9章2節 PID 制御モデルのモデリング

9. 2. 1. 速度制御のサンプルモデルを開いてみよう。

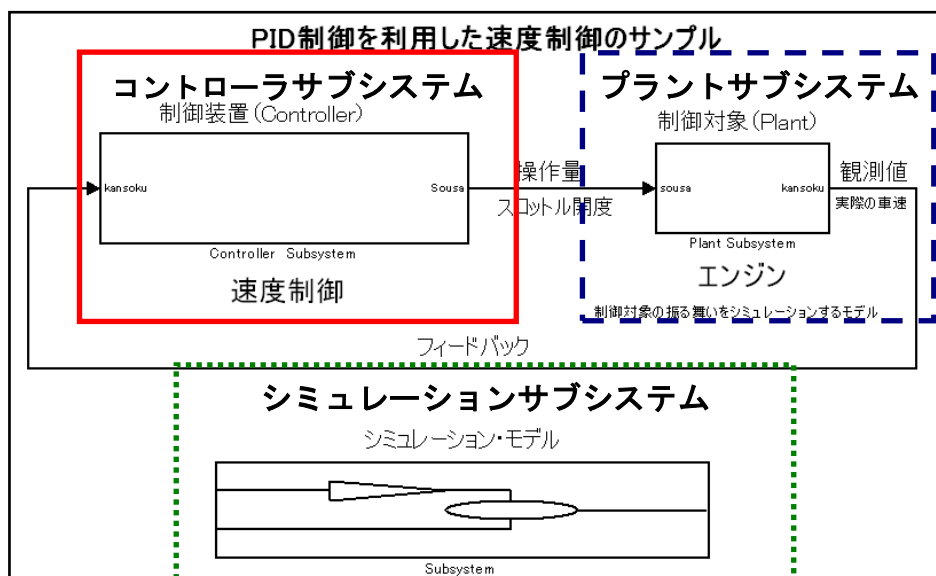
速度制御のサンプルモデルが入っているファイルを開いて見ましょう。

(添付の CD-R の[9 章 PID 制御モデルのモデリング¥ SpeedControl¥ベースモデル]フォルダ内から [Speed_Control.mdl]をデスクトップ上にコピーし、ファイルを開いて見ましょう。)



9. 2. 2. PID モデルの構成の説明

1. コントローラサブシステム、プラントサブシステム、シミュレーションサブシステムの3つから構成されています。
2. プラントサブシステムは制御対象であり、速度制御の例で言えばエンジンが該当します。これはエンジンの挙動をモデル化（模範した動き）した部分です。
3. コントローラサブシステムは、この場合、エンジンを制御する制御部分です。（ここの部分が開発対象）（例えば、速度制御の制御部分）
4. コントローラとプラントは次の信号で接続されています。
コントローラからプラントを操作する為の操作量（例えば、スロットル開度）
コントローラの入力となる観測値（例えば、エンジン回転数）
プラントからコントローラへのフィードバックである観測値（例えば、車速）
5. シミュレーションサブシステムは、検証目的に沿った入力変化のシナリオをコントローラへ入力し、コントローラの内部状態や出力、プラントの出力やプラントからのフィードバックなどを観測することにより、コントローラの挙動を検証する部分です。
ここでは、目標値が「0」から「100」に変化した場合のプラントの出力の変化を観測し、コントローラ（PID制御）の挙動を検証する為のシミュレーションがモデリングしてあります。（制限速度が60⇒100）



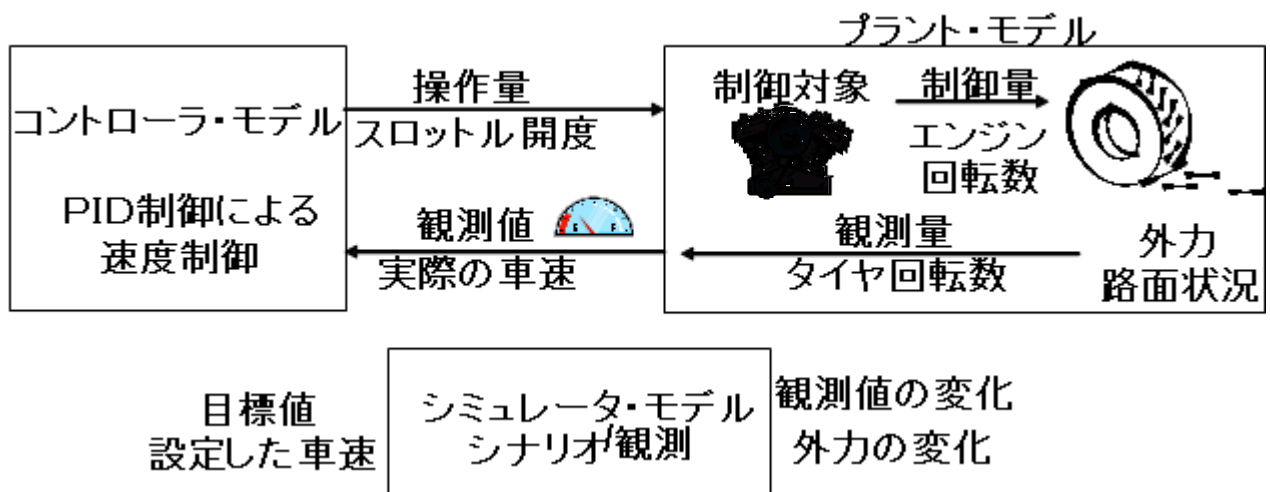
9. 2. 3. PID モデルのモデリング

速度制御に「PID 制御」を簡易的に適用した例

フィードバック制御に PID 制御を適用した速度制御

- ・ 目標値：設定した車速
- ・ 操作量：スロットル開度
- ・ 制御量：エンジン回転数
- ・ 外力、外乱：路面抵抗、勾配、風圧・・・
- ・ 観測値：実際の車速








9. 2. 4. コントローラとプラント



9章3節 PID 制御を実際にモデリングしてみよう。(課題4)

- ・コントローラサブシステムを開いて見ましょう。
- ・テキストにある PID モデルを見ながら、コントローラサブシステムを完成させましょう。

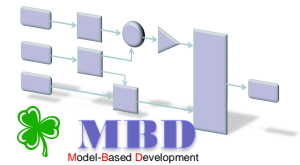
9. 3. 1. PID 制御モデルのライブラリとパラメータ

	Simulinkライブラリ	パラメータ
	Continuous/Integrator	
	Continuous/Derivative	
	MathOperations/Gain	ゲイン: $0.6 * 1.08$
	MathOperations/Gain	ゲイン: $1 / (0.5 * 10.8)$
	MathOperations/Gain	ゲイン: $0.125 * 10.8$
	MathOperations/Sum	符号リスト: +-
	MathOperations/Sum	符号リスト: +++

9. 3. 2. PID 制御モデルを検証してみよう。

- ・シミュレーションを実行し、シミュレーションサブシステムの **Scope** で実行結果を確認しましょう。

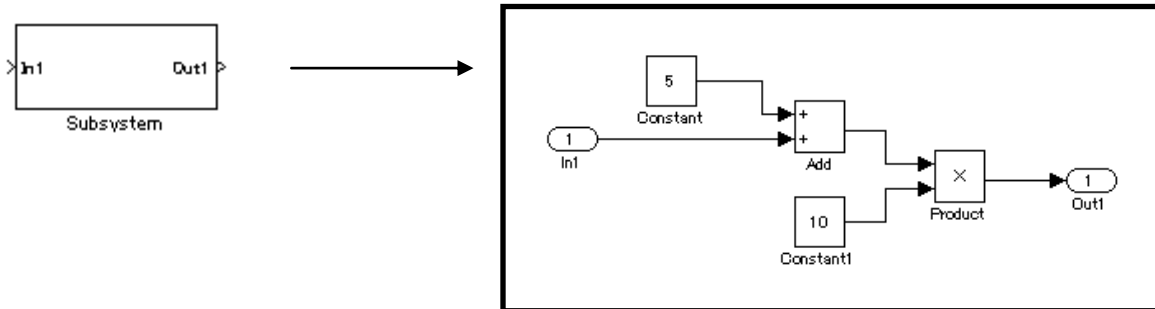
付録 よく使われるブロック



Subsystem

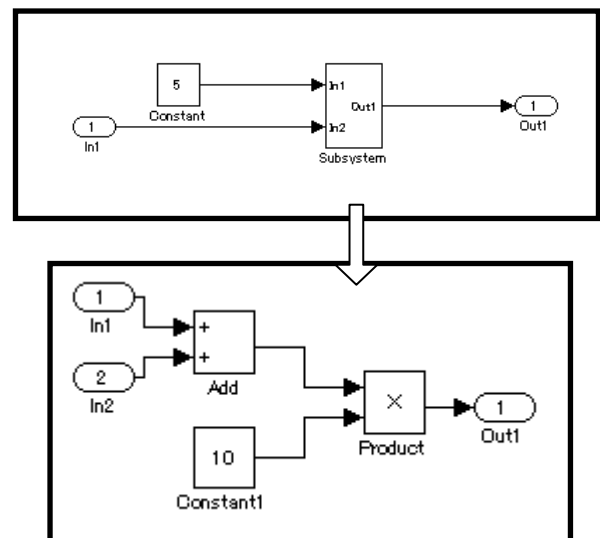
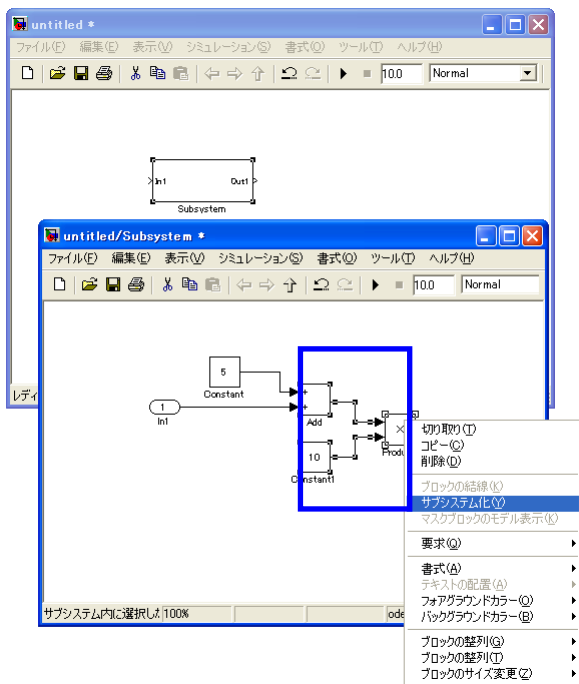
目的 システムを1つのブロックとしてサブシステム化することができます。

ライブラリ Ports & Subsystems



Subsystem ブロックをダブルクリックし、入力端子(In1)と出力端子(Out1)の間にモデルを作成することでサブシステム化することができます。

また、モデル作成後にモデルの一部をサブシステム化したい場合には、サブシステム化したいモデル範囲を指定して右クリックし、サブシステム化することができます。



モデル名 : Subsystem.mdl

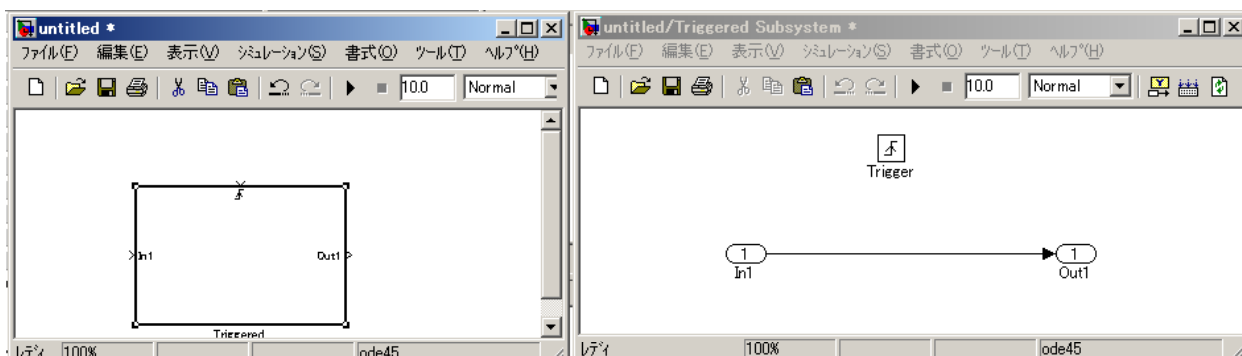
◆条件付き実行のサブシステム(イベントドリブンシステム)

1、Triggered サブシステム

イベント信号が発生する度に実行されるサブシステムです。

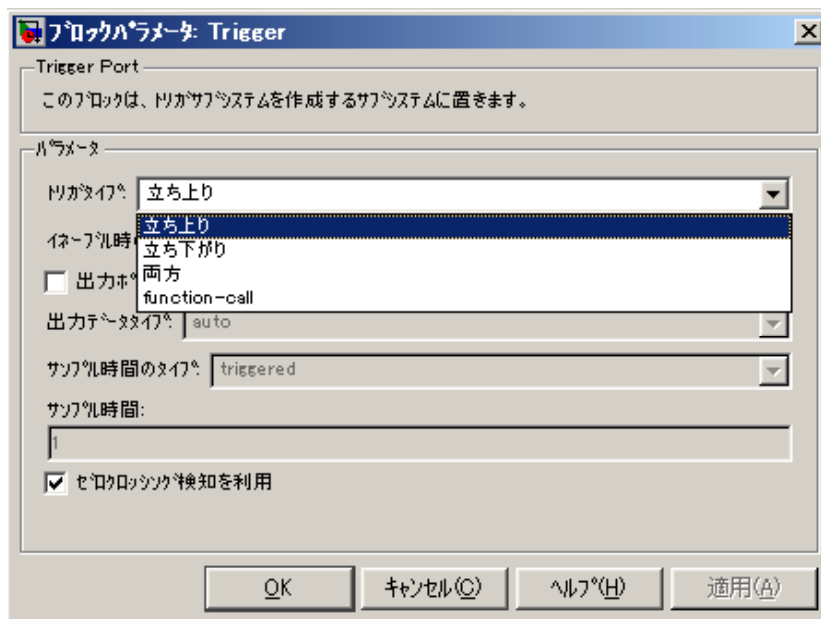
まずは、左下の Triggerd Subsystem ブロックを使います。

ブロックをダブルクリックすると下右側のサブシステムが開きます。

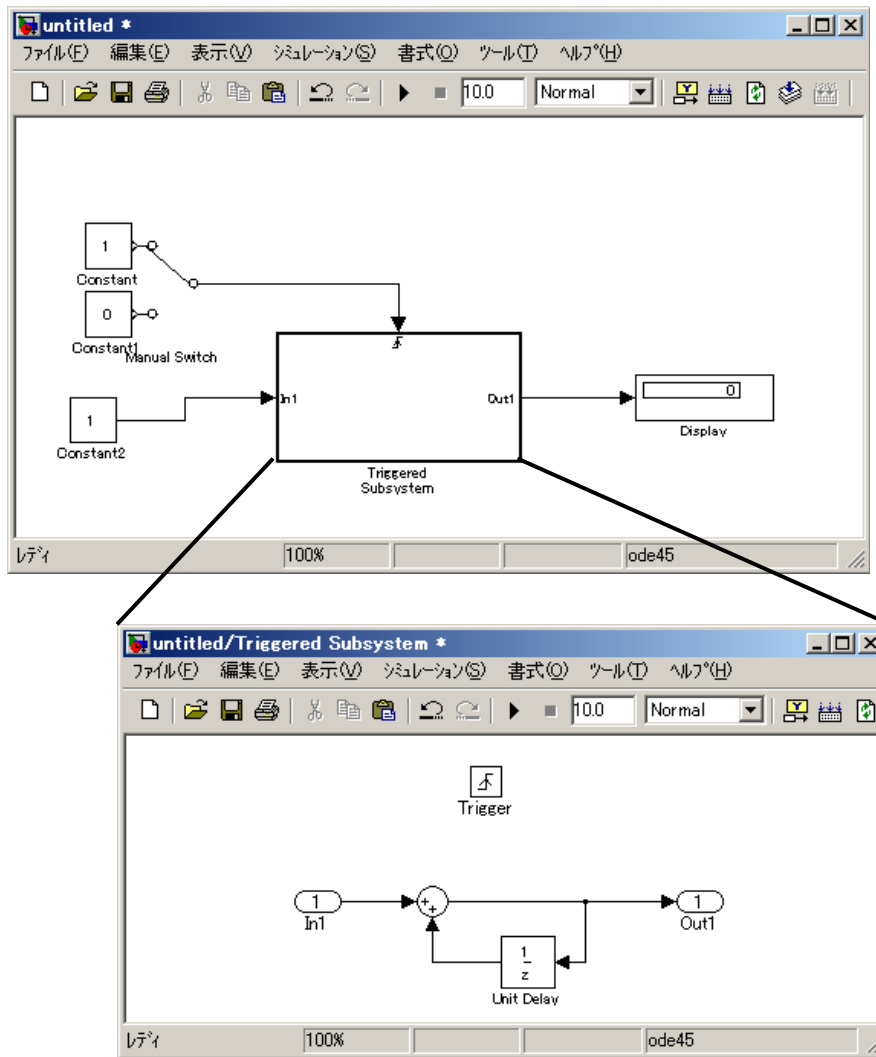


Trigger をダブルクリックするとブロックパラメータが開きます。

サブシステムが作動する条件をトリガタイプで、選択できます。



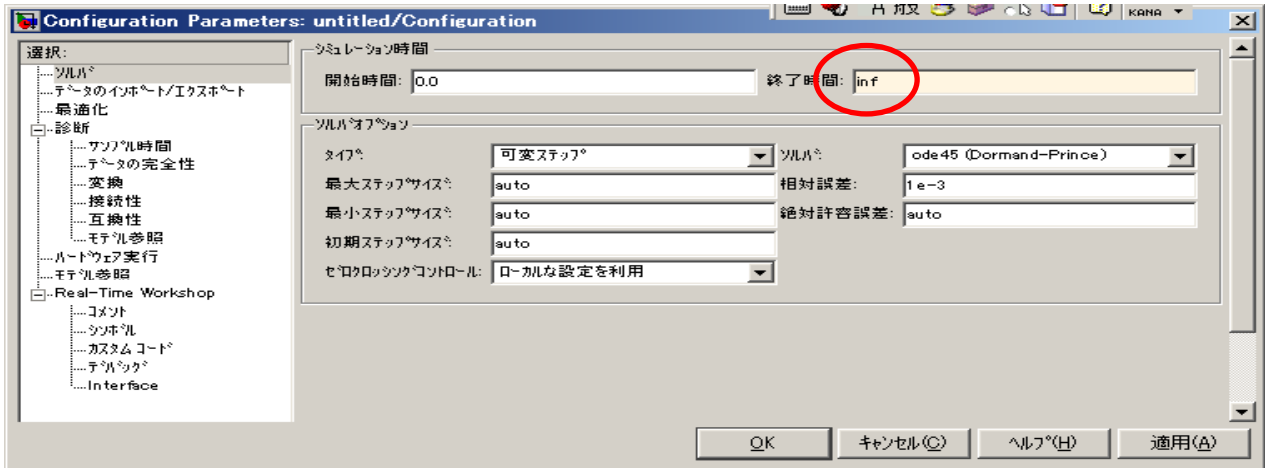
簡単な例を実際を作って動作させましょう。



使用ブロックは、constant, Manual Switch, Display, Unit Delay です。

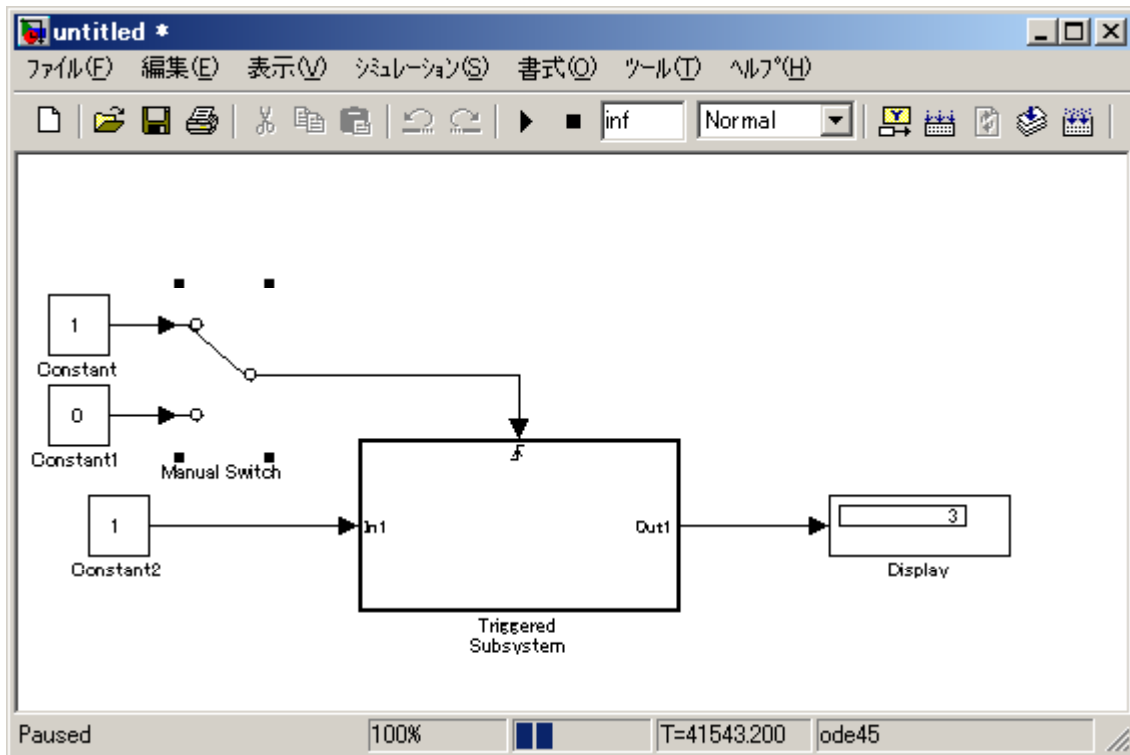
Triggered サブシステムは、時間に非同期の離散システムなので、サブシステム内には、サンプル時間を持たないブロックしか置く事ができません。そこで、Unit Delay ブロックのサンプル時間には、「-1」を入力し、それ自身では、サンプル時間を持たない設定にします。また、トリガーイベントは、立ち上がりに設定してください。Unit Delay は、ブロックによって、1 回前の動作した出力値が保存されます。

シミュレーションパラメータを以下のようにして、シミュレーションを実行します。
終了時間の「inf」は、シミュレーションの stop ボタンをクリックするまで続く設定になります。



シミュレーション結果は、マニュアルスイッチをダブルクリックして、1 になった回数をカウントします。

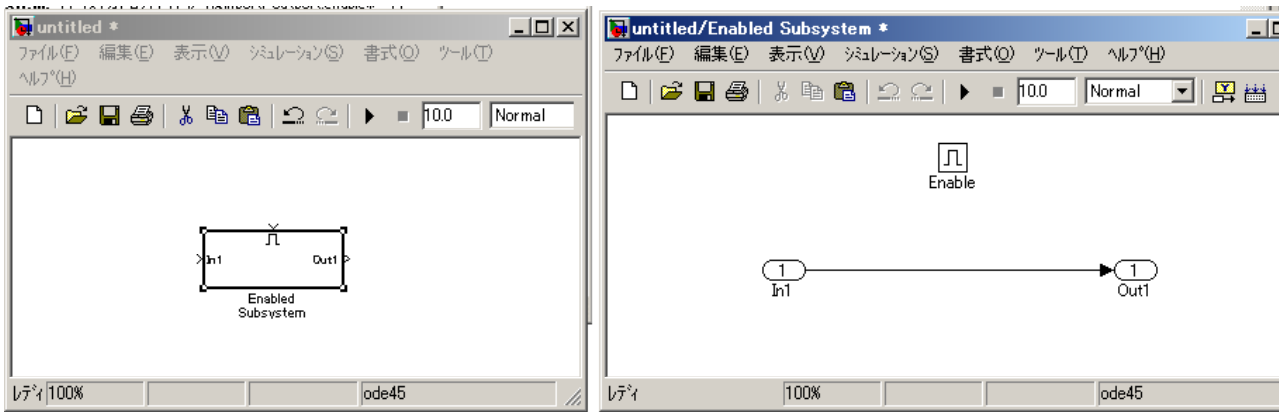
イベントの発生回数が表示されることとなります。



2、Enabled サブシステム

イベント信号が0より大きい場合に実行されるサブシステムです。

下のブロックを用いて作成します。

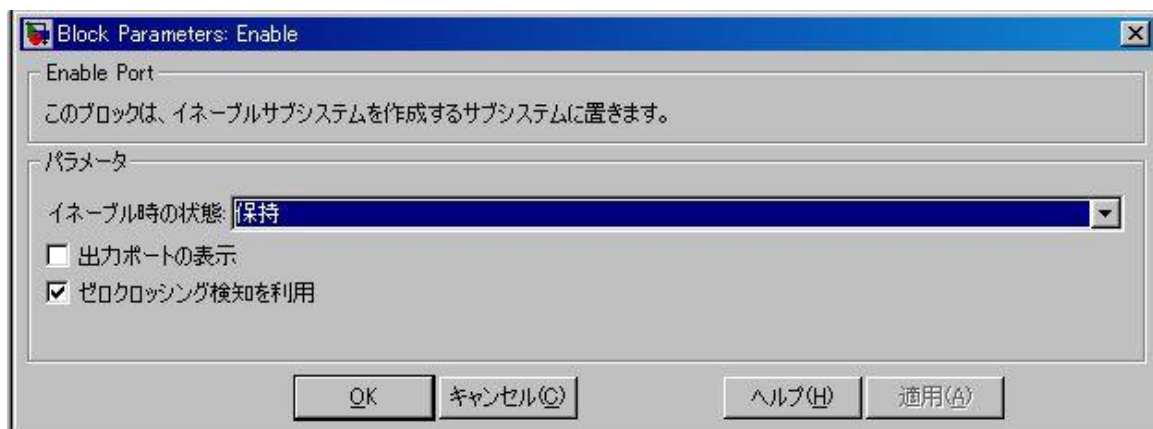


このサブシステムでは、再度動作した時に、サブシステム内の状態量を持つブロック（例えば、Integrator ブロック）の初期状態の扱いを2種類選択することができます。

ブロックパラメータのイネーブル時の状態設定でそれぞれが以下のようにになります。

保持： 初期状態は、前回動作したときの状態をもつブロックの最後の値を使用

reset： 初期状態は、状態量をもつブロックで指定した初期値を使用

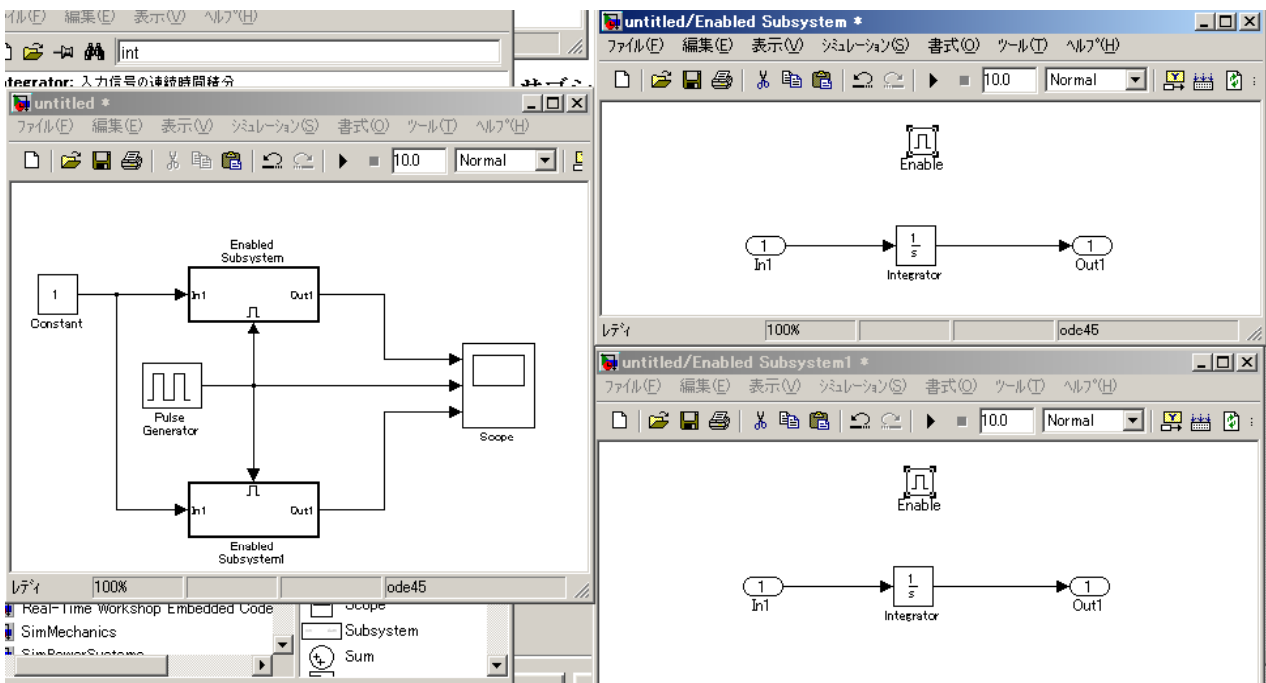


Enable ブロックのパラメータの設定違いによる挙動を比較します。

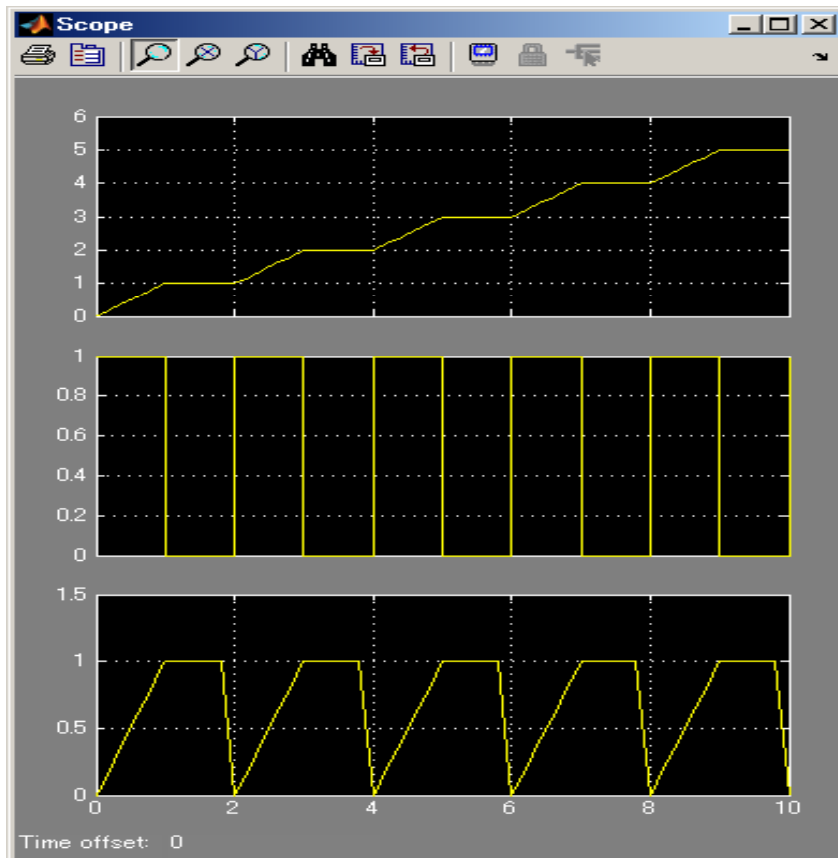
モデル名 : enabl.mdl

必要なブロックは、Enable Subsystem×2、Constant、Pulse Generator(周期 2sec)、Scope
です。

上の Enable ブロックは、保持にし、下の Enable ブロックは、リセットにします。
スコープはブロックパラメータを開いて、3軸で信号が見られるように設定し、また、
サブシステム内部に積分器を挿入します。



モデルが作成できたら、シミュレーションしてみましょう。
どんな波形になったでしょうか？



イベント信号が **1** の間は動作し、**0** の間は動作していない様子が確認できます。
なお、「保持」の場合は、動作していない状態から動作する状態になる場合（例えば、2秒）に、サブシステム内の Integrator ブロックの初期状態は前回動作した Integrator ブロックの最後の値を使用して計算が始まります。

それに比べ「リセット」の場合は、サブシステム内の Integrator ブロックの初期状態が Integrator ブロックで指定した初期状態の値（デフォルト値：0）を使用して計算が始まります。

Logical Operator

目的 論理演算を行います。

ライブラリ **Logic and Bit Operations**



Logical Operator ブロックは、そのブロックの入力に対して、指定した論理演算を実行します。入力値が非ゼロの場合は **TRUE** (1) に、ゼロの場合は **FALSE** (0) になります。

Operator パラメータリストを用いて入力を接続する **Boolean** 演算を選択してください。このブロックは、選択した演算子を表示するように更新を受けます。サポートされる演算を以下に示します。

演算詳細

表 1 ●AND 演算の真理値表

a の値	b の値	a AND b の演算結果
1 (真)	1 (真)	1 (真)
0 (偽)	1 (真)	0 (偽)
1 (真)	0 (偽)	0 (偽)
0 (偽)	0 (偽)	0 (偽)

表 2 ●OR 演算の真理値表

a の値	b の値	a OR b の演算結果
1 (真)	1 (真)	1 (真)
0 (偽)	1 (真)	1 (真)
1 (真)	0 (偽)	1 (真)
0 (偽)	0 (偽)	0 (偽)

表 3 ●XOR 演算の真理値表

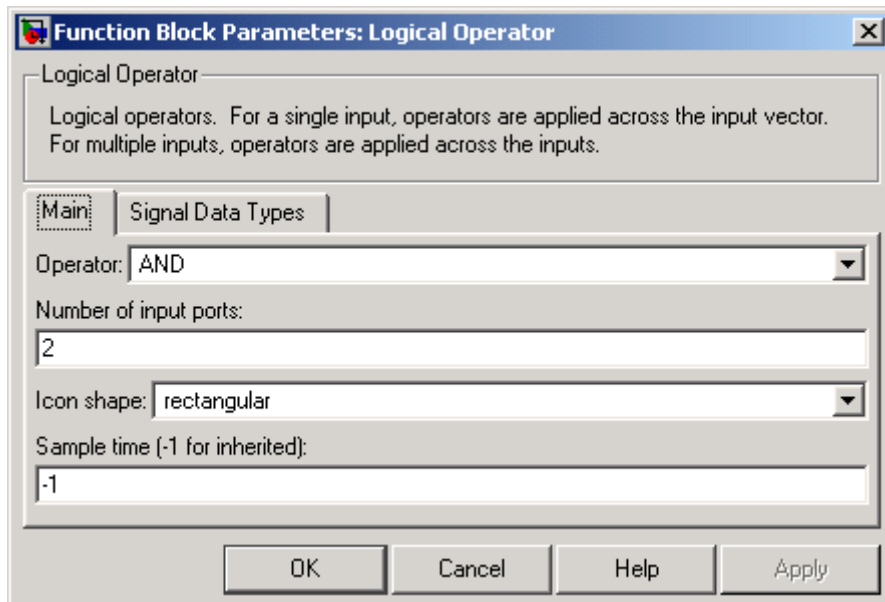
a の値	b の値	a XOR b の演算結果
1 (真)	1 (真)	0 (偽)
0 (偽)	1 (真)	1 (真)
1 (真)	0 (偽)	1 (真)
0 (偽)	0 (偽)	0 (偽)

表 4 ●NOT 演算の真理値表

a の値	NOT a の演算結果
1 (真)	0 (偽)
0 (偽)	1 (真)

- AND** :すべての入力が **TRUE** の場合は **TRUE**
- OR** :1つ以上の入力が **TRUE** の場合は **TRUE**
- NAND** :1つ以上の入力が **FALSE** の場合は **TRUE**
- NOR** :いずれの入力も **TRUE** でない場合は **TRUE**
- XOR** :奇数個の入力が **TRUE** の場合は **TRUE**
- NOT** :入力が **FALSE** の場合は **TRUE**

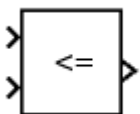
入力ポートの数は、Number of input ports パラメータで指定します。出力タイプは、Output data type mode パラメータと Output data type パラメータの両方または、どちらか一方で指定します。TRUE の場合は出力値が 1 に、FALSE の場合は 0 になります。



Relational Operator

目的 比較演算を行います。

ライブラリ **Logic and Bit Operations**



Relational Operator ブロックは、2つの入力に対して指定した比較演算を実行します。比較演算子パラメータによって、2つの入力をつなぐ比較演算子を選択します。ブロックは、選択した演算子を表示するように更新されます。

サポートされる演算を以下に示します。

演算子	説明
==	最初の入力が2番目の入力と等しければ真
~=	最初の入力が2番目の入力と等しくなければ真
<	最初の入力が2番目の入力より小さければ真
<=	最初の入力が2番目の入力以下であれば真
>=	最初の入力が2番目の入力以上であれば真
>	最初の入力が2番目の入力より大きければ真

入力は、スカラ、配列、またはスカラと配列の組合せて指定することができます。スカラ入力の場合、出力はスカラです。

配列入力の場合、出力は同じ大きさの配列です。この場合の各要素は、入力配列の要素単位の比較の結果となります。スカラと配列が混在する入力の場合、出力は配列です。この場合の各要素は、スカラと対応する配列要素との間の比較の結果です。

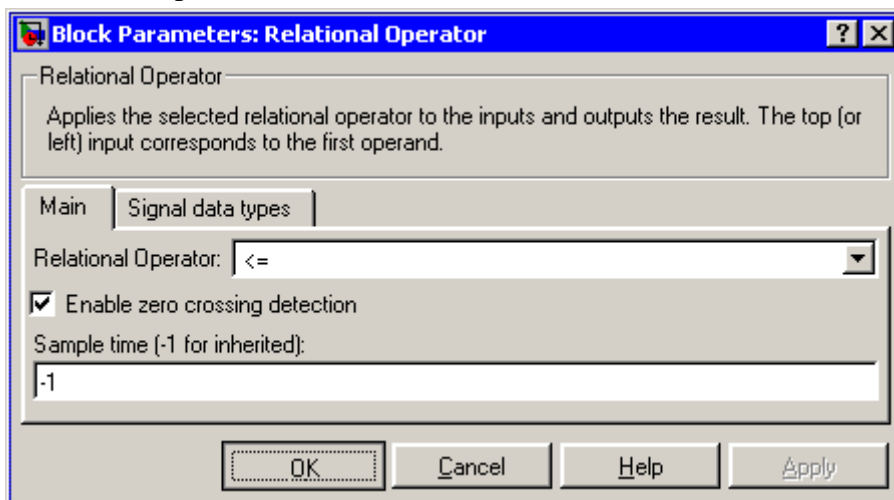
サポートされるデータタイプ

Relational Operator ブロックは、Simulink がサポートするすべてデータタイプ(固定小数点データタイプを含む)の実数および複素数信号を受け入れます。しかし、出力データタイプモード パラメータは Logical に設定された場合、入力は Boolean あるいは double だけです。演算子が == あるいは != の場合、1 つの入力が実数でもう一方が複素数でもかまいません。

Simulink でサポートされるデータタイプについては、Using Simulink ドキュメンテーションの”Simulink でサポートされるデータタイプ”を参照してください。

パラメータとダイアログボックス

Relational Operator ブロックのメインペインを下に示します。



比較演算子

2つの入力を比較するために使用する、比較演算子を指定します。

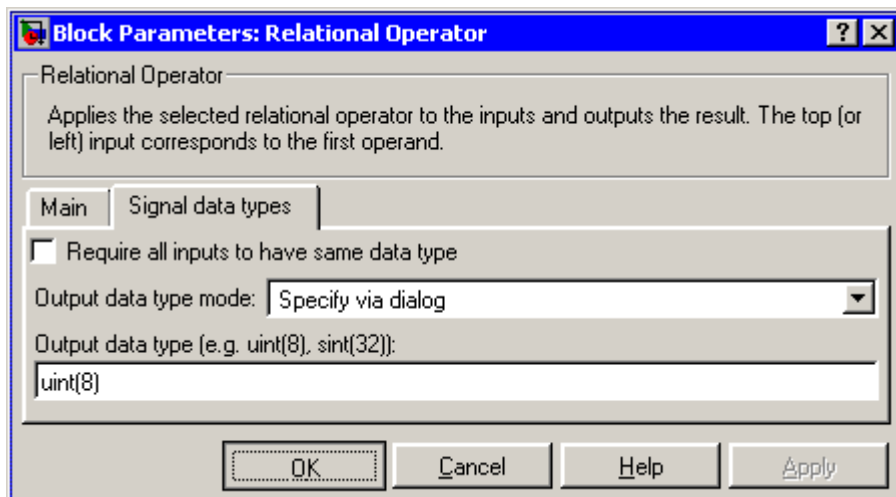
ゼロクロッシング検知の利用 選択すると、ゼロクロッシング検出を有効にします。

詳細は、Using Simulink ドキュメンテーションの ”ゼロクロッシング検出” を参照してください。

サンプル時間(継承は-1)

サンプル間の時間間隔を指定します。サンプル時間を継承するには、このパラメータを-1に設定します。詳細は、オンラインマニュアルの”サンプル時間の指定”を参照して下さい。

Relational Operator ブロックの信号のデータタイプのペインは、次のように表示されます。



すべての入力値が同じデータタイプを持つ 選択すると、入力が同じデータタイプを持ちます。

出力データタイプモード

出力データタイプを **Boolean** に設定するか、あるいはデータタイプを出力データパラメータの中から選択します。

もう一つの方法として、**Configuration Parameters** ダイアログボックスのシミュレーションとコード生成 最適化ペインで、**Logical** を選択すると出力データタイプをブーリアンデータとして論理信号を使用します。パラメータに決めさせることもできます。

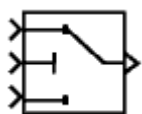
Logical を選択して **Boolean** 論理信号 (**Boolean Logic Signals**) を有効にしてある場合、出力データタイプは常に **Boolean** となります。**Logical** を選択してブーリアンデータとしての論理信号の使用を無効にしてある場合、出力データタイプは入力データタイプと一致します。このタイプは常に **double** です。

出力データタイプ

出力データタイプを指定します。厳密に 0 を示すデータタイプだけを使用する必要があります。この条件を満たすデータタイプには、符号つきおよび符号無しの整数と任意の浮動小数点データタイプを含みます。出力データタイプモード パラメータに対する「ダイアログにより指定」を選択した時にだけ、このパラメータが表示されます。

Switch

目的 第2の入力の値に応じて最初の入力と3番目の入力の出力を切り替えます。
ライブラリ **Signal Routing**

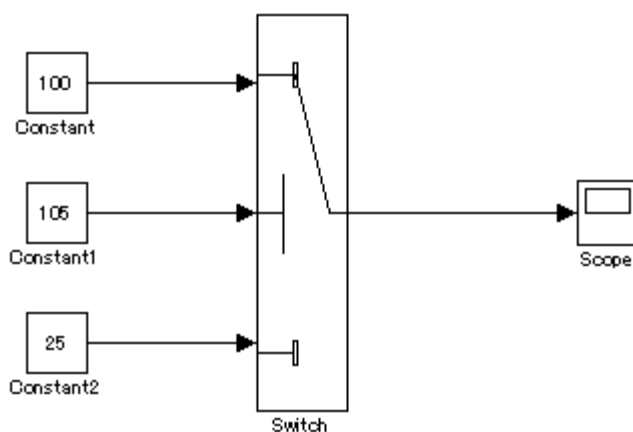


Switch ブロックは、2番目（中央）の値に応じて、最初（上）の入力または3番目（下）の入力を通過させます。最初と3番目の入力をデータ入力といいます。2番目の入力を制御入力といいます。

最初の入力が通過する条件 パラメータを使って、入力を通過させる条件を選択できます。制御入力が閾値以上かどうか、閾値よりも大きいかどうか、またはゼロ以外かどうかをブロックでチェックすることができます。制御入力が最初の入力が通過する条件で設定された条件に一致する場合は、最初の入力が渡されます。

そうでない場合、3番目の入力が渡されます。

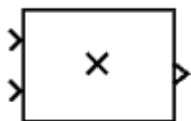
モデル名 : switchblockmdl.mdl



Product

目的 入力を乗算または除算します。

ライブラリ **Math Operations**



Product ブロックは、そのブロックの入力に対して乗算または除算を実行します。

このブロックは、要素に関する乗算または行列の乗算を用いて出力を生成します。どちらを使用するかは **乗算** パラメータの値によって決まります。入力数 パラメータを用いて演算を指定してください。乗算 (*) キャラクタと除算 (/) キャラクタは、入力に関して実行する演算を指示します。

2つ以上の入力が存在する場合、キャラクタの数は入力の数に等しくなければなりません。たとえば、“*/*” は3つの入力を必要とします。この例の場合、乗算 パラメータを **Element-wise** (要素単位) に設定すると、このブロックは最初(一番上)の入力の要素を2番目(中央)の入力の要素で除算してから、3番目(一番下)の入力の要素を乗算します。

このケースの場合、このブロックへのすべての非スカラー入力は同じ次元を持たなければなりません。ただし、乗算 パラメータを **Matrix** に設定した場合、このブロックは、入力の行列積 “*” と入力の逆 “/” を出力します。ここで、演算の順序は、入力数・パラメータの入力値に従います。入力の次元は、行列積が定義されるように指定しなければなりません。

注意: 入力ベクトルに関して点乗積を実行するには **Dot Product** ブロックを使用します。

入力の乗算のみが必要な場合、入力の数に等しい数値パラメータ値が “*” キャラクタの代わりに提供されます。これは、要素に関する乗算または行列の乗算と併用できます。

単一のベクトルが入力で、乗算 パラメータを **Element-wise** に設定した場合、単一の “*” がこのブロックにベクトル要素のスカラー積を出力させます。単一の “/” は、このブロックにベクトル要素のスカラー積の逆を出力させます。

単一の行列が入力で、乗算 パラメータを **Element-wise** に設定した場合、単一の “*” または “/” がこのブロックにエラーを出力させます。ただし、乗算 パラメータを **Matrix** に設定した場合、単一の “*” は、このブロックに未変更の行列を出力させます。また、単一の “/” は、このブロックに行列の逆を出力させます。

Product ブロックは、まず指定した乗算または除算演算を入力に対して実行してから、指定した丸めおよびオーバーフローモードを用いて結果を出力データタイプに変換します。

入力数

入力の数または “*” 記号と “/” 記号の組み合わせを入力します。

このパラメータについての詳細は、上記の”説明”を参照してください。

乗算

要素に関する乗算または行列の乗算を指定します。このパラメータについての詳細は、上記の「説明」を参照してください。

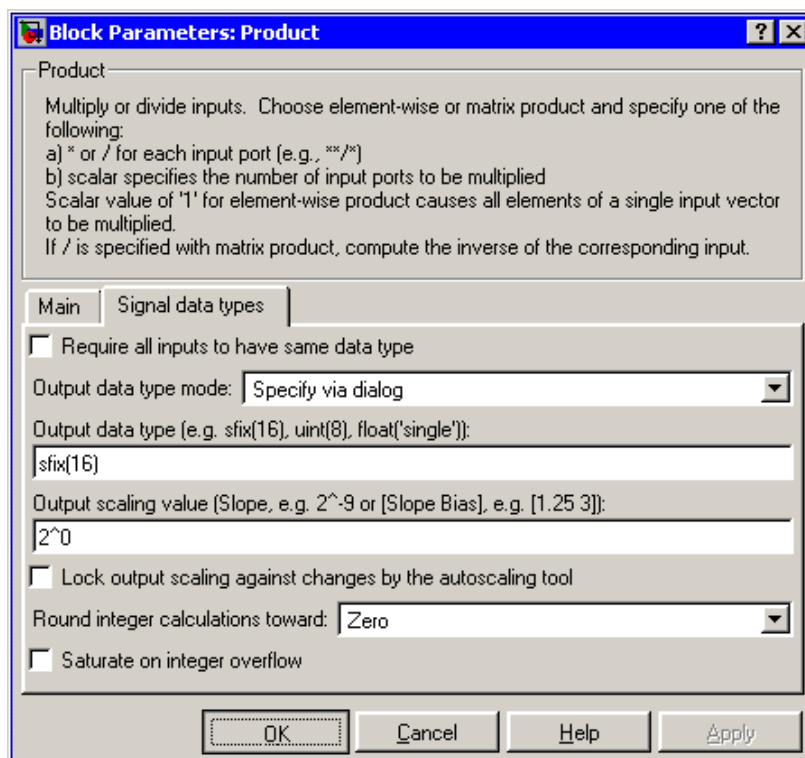
サンプル時間(継承は-1)

サンプル間の時間間隔を指定します。

サンプル時間を継承するには、このパラメータを -1 に設定します。

詳しくは、オンラインマニュアルの”サンプル時間の指定”を参照してください。

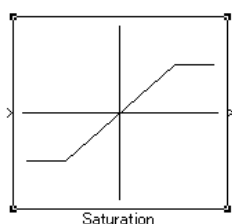
Product ブロックダイアログの信号のデータタイプペインは、以下のように表示されます。



Saturation

目的 飽和要素を表現します。

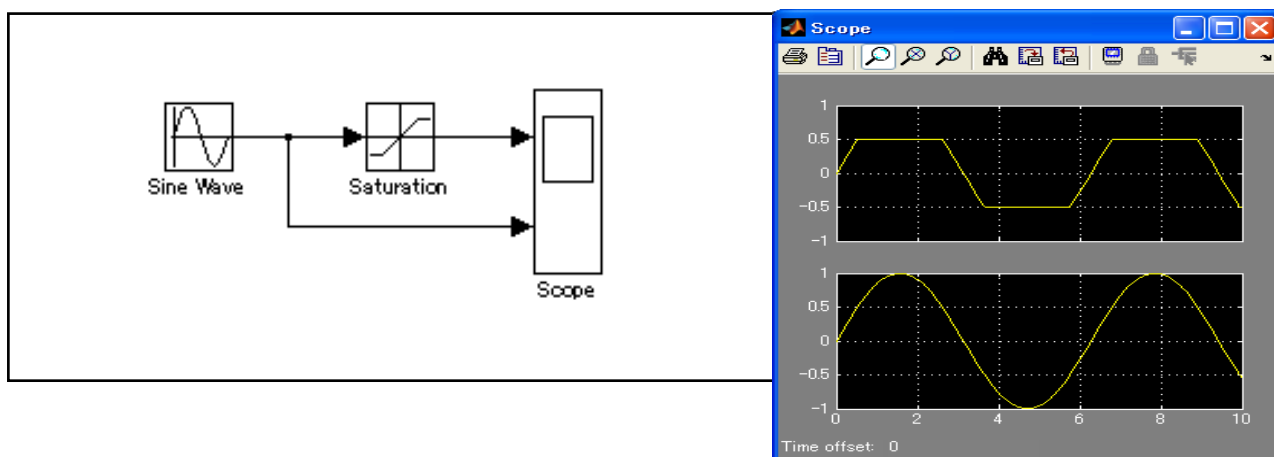
ライブラリ **Discontinuities**



Saturation ブロックは、そのブロックの入力信号が上限値より大きな値の場合には、上限値を出力し、下限値より小さい値の場合には、下限値を出力します。



モデル名 : Saturation.mdl



このモデルでは、上限値 0.5 下限値 -0.5 と設定した場合の Saturation ブロック適用による波形の変化を示しています。

Ground

目的 未接続の入力端子を固定し、ワーニングを防ぎます。

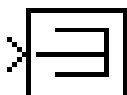
ライブラリ Sources



Terminator

目的 未接続の出力端子を終端させます。

ライブラリ Sinks



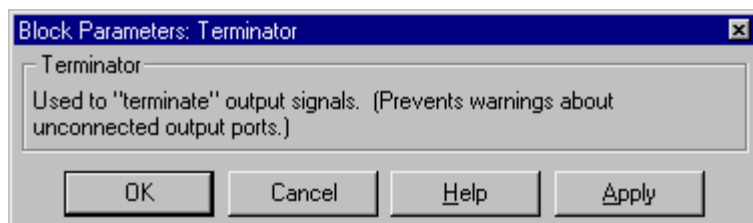
Terminator ブロックは、出力端子が他のブロックに接続されていないブロックを終端させるのに使用することができます。未接続の出力端子をもつブロックを用いてシミュレーションを実行すると、Simulink はワーニングメッセージを表示します。Terminator ブロックを使ってこれらのブロックを終端させると、ワーニングメッセージを回避することができます。

サポートされるデータタイプ

Terminator ブロックは、固定小数点データタイプなど、Simulink がサポートする任意データタイプの実数値信号または複素数信号を受け入れます。

Simulink がサポートするデータタイプについては、Using Simulink ドキュメンテーションの“Simulink でサポートされるデータタイプ”を参照してください。

パラメータとダイアログボックス



Goto

目的 ブロック入力を From ブロックに渡します。

ライブラリ **Signal Routing**

説明



Goto ブロックは、該当する From ブロックに入力を渡します。

入力は、任意のデータタイプの実数値または複素数値信号またはベクトルです。

From ブロックと Goto ブロックを使用すると、それらを実際に接続しなくても、ブロックから別のブロックへ信号を渡すことができます。

Goto ブロックは複数の From ブロックに入力を渡すことができますが、From ブロックは1つの Goto ブロックからしか信号を受け取ることができません。しかし、その Goto ブロックへの入力は、ブロックが物理的に接続されているかのように、それに関連付けられた From ブロックに渡されます。

Goto ブロックと From ブロックは、タグパラメータで定義された Goto タグの使用によって対応付けられます。

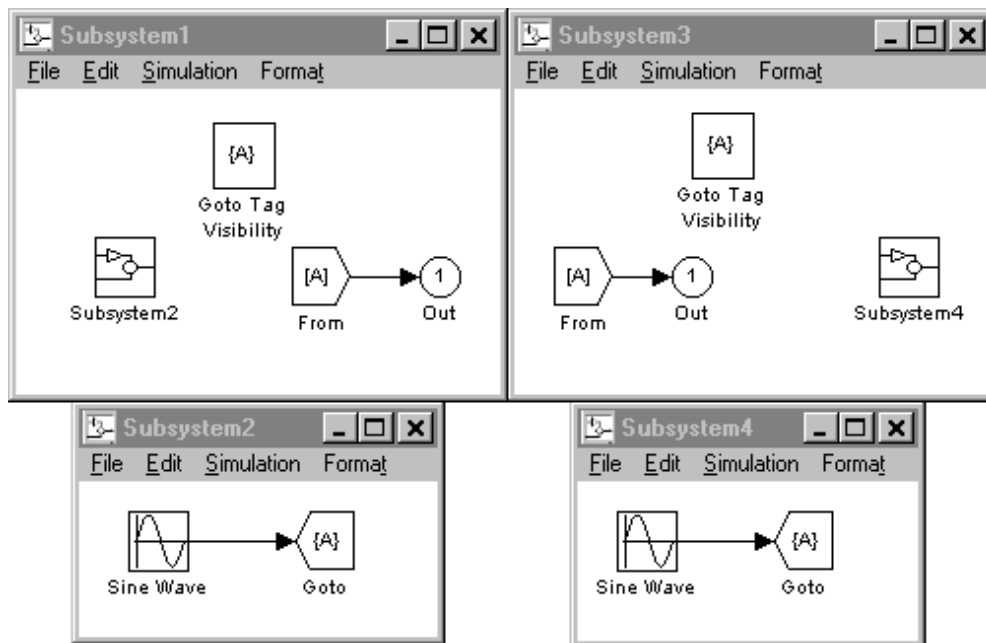
同じタグ名を使用する Goto ブロックと From ブロックが同じサブシステムにある場合は、local タグを使用してください。

同じタグ名を使用する Goto ブロックと From ブロックが異なるサブシステムにある場合は、global または scoped タグを使用する必要があります。

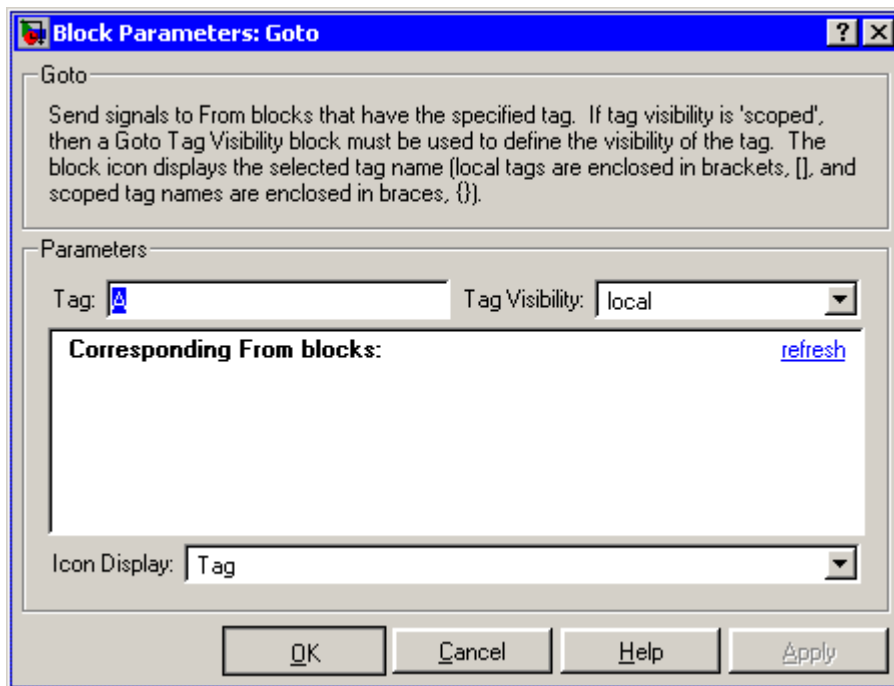
タグを global として定義した場合、そのタグの使用はすべて、同じ信号にアクセスします。

scoped として定義されたタグは、モデル内の複数の場所で使用することができます。

つぎの例は、同じ名前 (A) の 2 つの scoped タグを使用するモデルを示しています。



パラメータとダイアログボックス



タグ

Goto ブロック識別子。

このパラメータは、範囲がこのブロックで定義される Goto ブロックを識別します。

タグの表示

Goto ブロックタグの範囲 (local、scoped、または global)。

デフォルトは local です。

Corresponding From blocks

この Goto ブロックに接続された From ブロックのリスト。

このリスト内のエントリをダブルクリックすると、対応する From ブロックが強調表示されます。

アイコンの表示

ブロックのアイコンに表示するテキストを指定します。

オプションは、ブロックのタグ、ブロックが表す信号の名前、またはタグと信号名の両方です。

モデル名 : goto_from.mdl

From

目的 Goto ブロックから入力を受け入れます。

ライブラリ **Signal Routing**



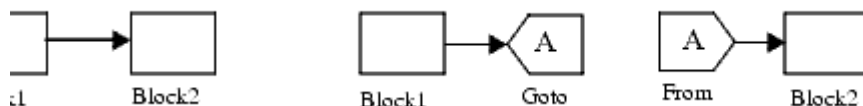
From ブロックは、対応する Goto ブロックから信号を受け入れて、出力として渡します。出力のデータタイプは、Goto ブロックからの入力のデータタイプと同じです。

From ブロックと Goto ブロックを使用すると、それらを実際に接続しなくても、ブロックから別のブロックへ信号を渡すことができます。Goto ブロックを From ブロックに関連付けるには、Goto タグ パラメータに Goto ブロックのタグを入力します。

From ブロックは 1 つの Goto ブロックからしか信号を受け取ることができませんが、Goto ブロックは複数の From ブロックに信号を渡すことができます。

次の図は、Goto ブロックと From ブロックの使用が、それらのブロックに接続されているブロックを接続することに等しいことを示しています。左のモデルでは、Block1 は Block2 に信号を渡します。このモデルは、右のモデルと同じです。

右のモデルでは、Block1 を Goto ブロックに接続して、その信号を From ブロックに渡し、そこから Block2 に渡します。



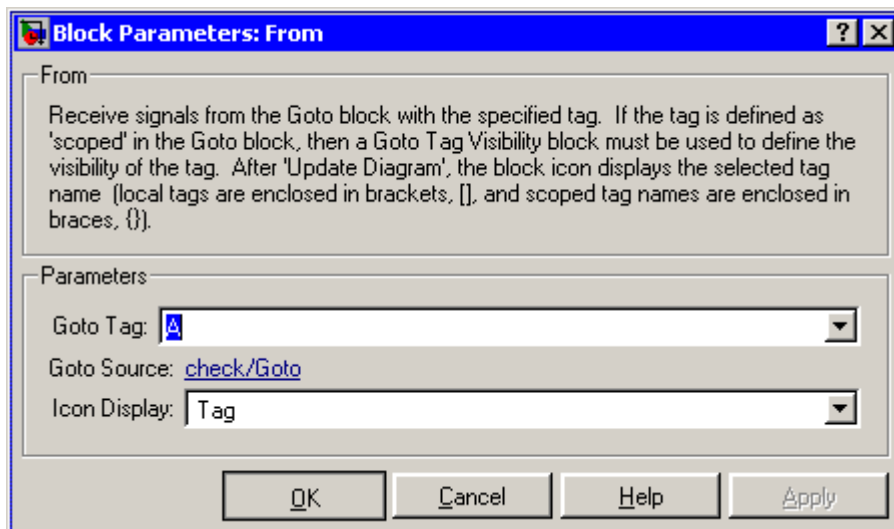
Goto ブロックのタグの可視性によって、その信号を受け取ることができる From ブロックが決まります。詳細については、Goto と Goto Tag Visibility を参照してください。このブロックは、Goto ブロックのタグの可視性を示します。

ローカルタグ名は、角括弧 ([]) で囲まれます。

範囲を限定されたタグ名は、中括弧 ({}) で囲まれます。

グローバルタグ名は、追加の文字なしで表示されます。

パラメータとダイアログボックス



Goto タグ

この From ブロックに信号を渡す Goto ブロックのタグ

Goto ソース

この From ブロックに接続された Goto ブロックのパス。
パスをダブルクリックすると、Goto ブロックが表示されて、強調表示されます。

アイコンの表示

From ブロックのアイコンに表示するテキストを指定します。
オプションは、ブロックのタグ、ブロックが表す信号の名前、またはタグと信号名の両方です。

Bus Creator

目的 信号バスを作成します。

ライブラリ **Signal Routing**



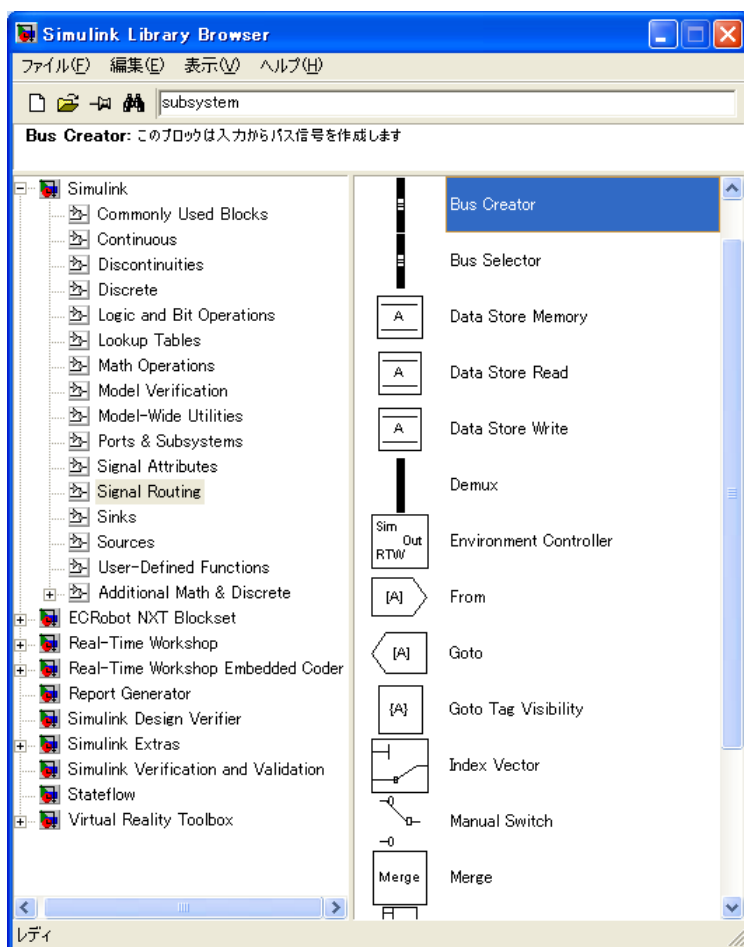
Bus Selector

目的 入力バスから入ってくる信号を選択します。

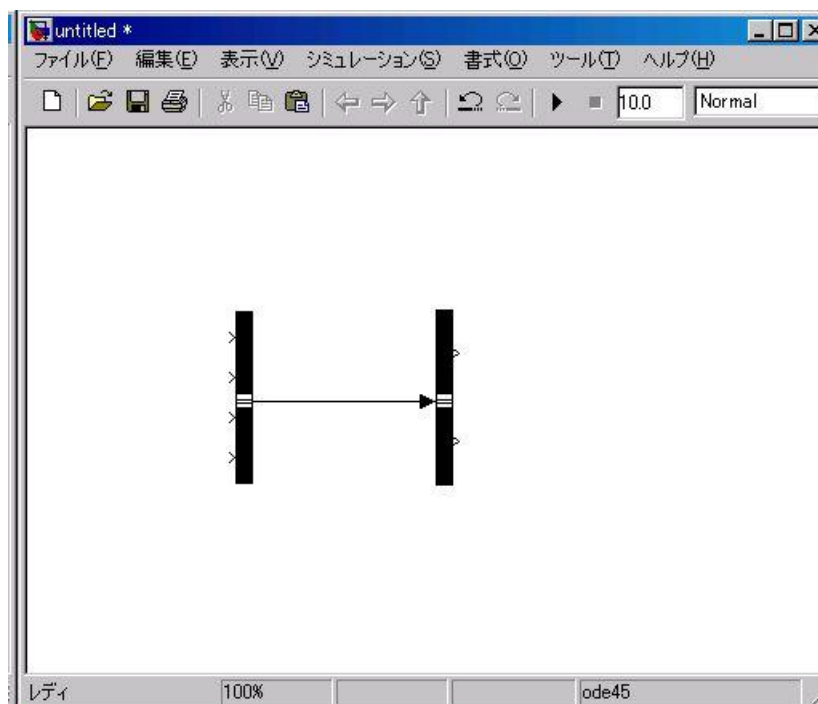
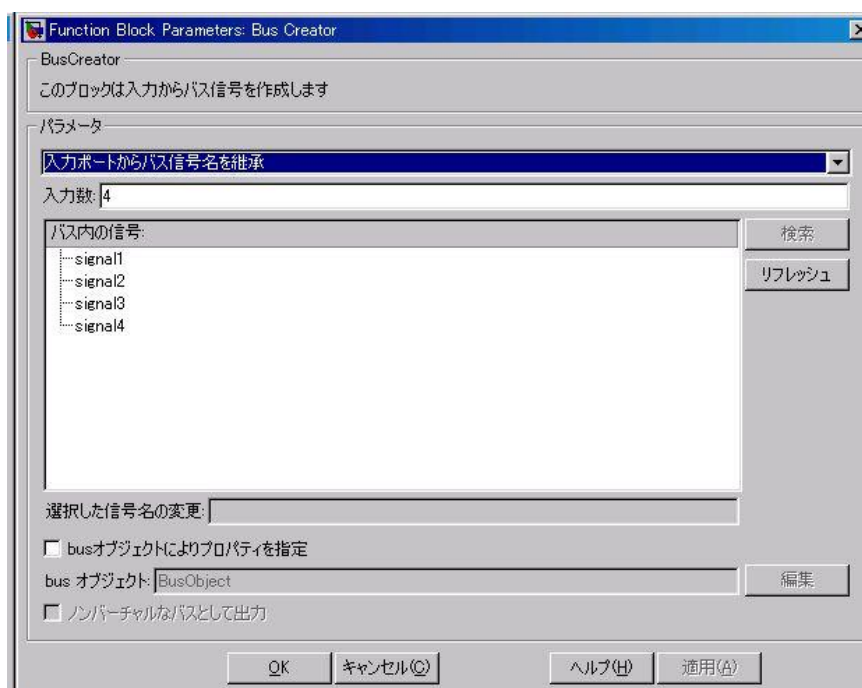
ライブラリ **Signal Routing**



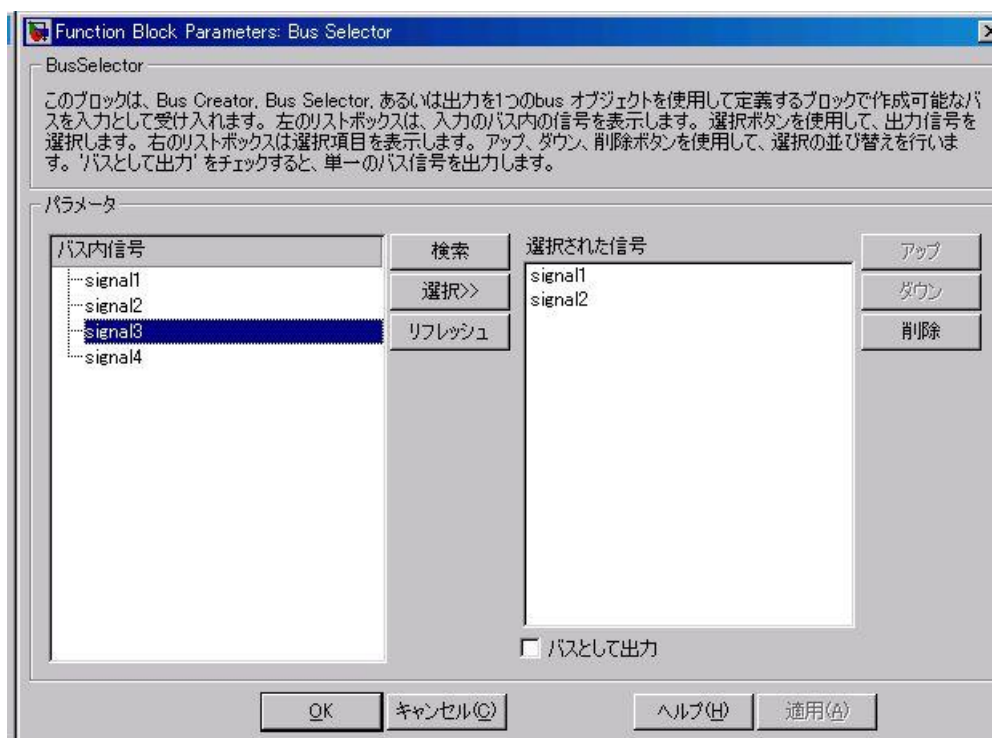
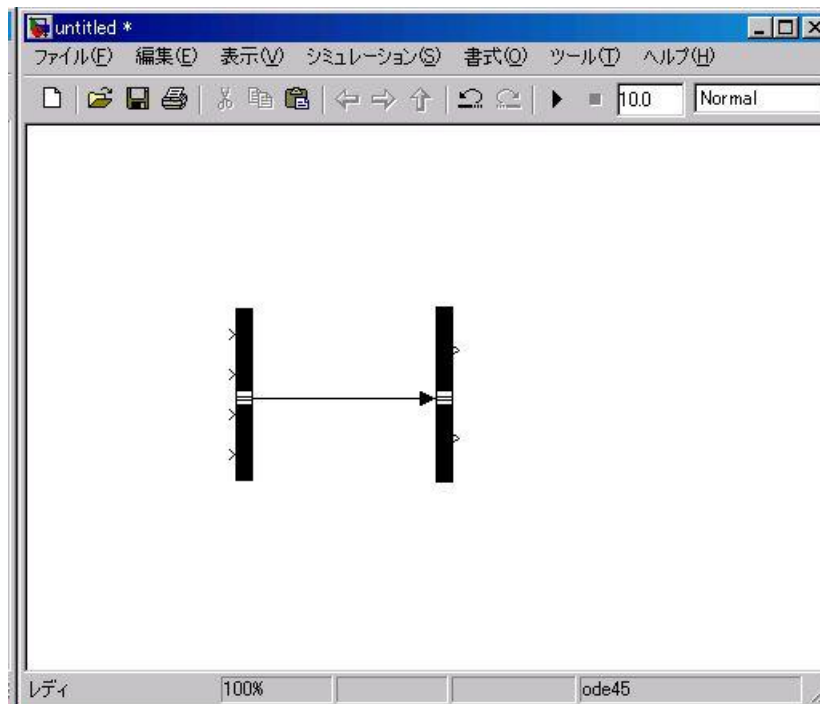
Bus Creator をドラッグ&ドロップして、Bus Creator をダブルクリックするとブロックパラメータが開くので、入力数を記入します。



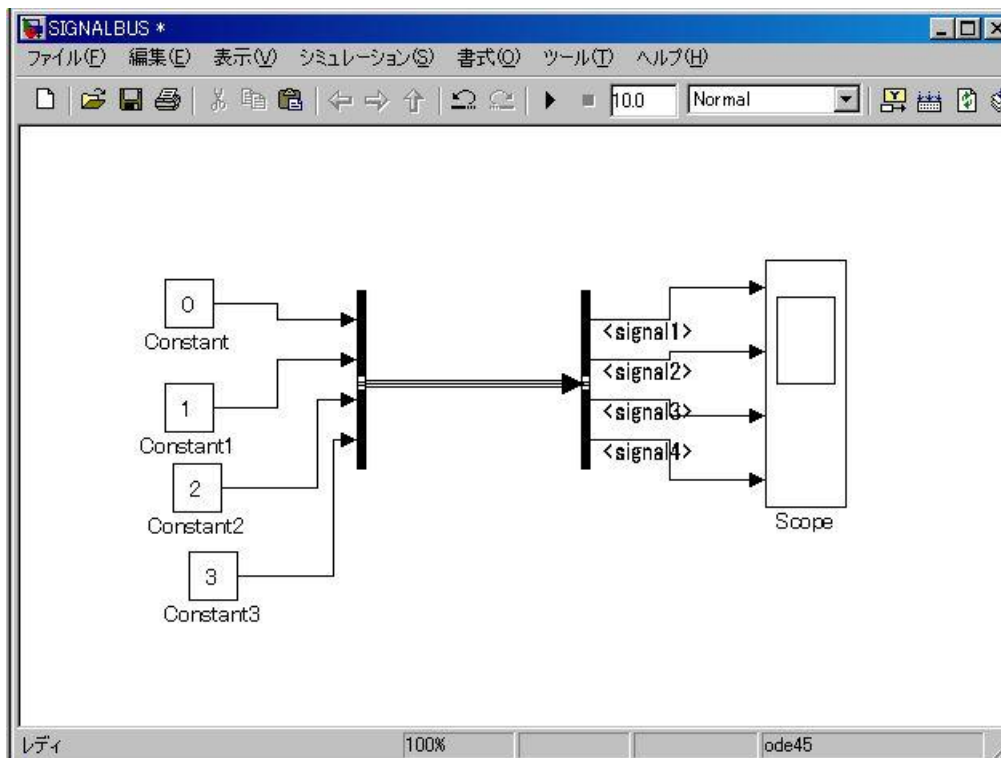
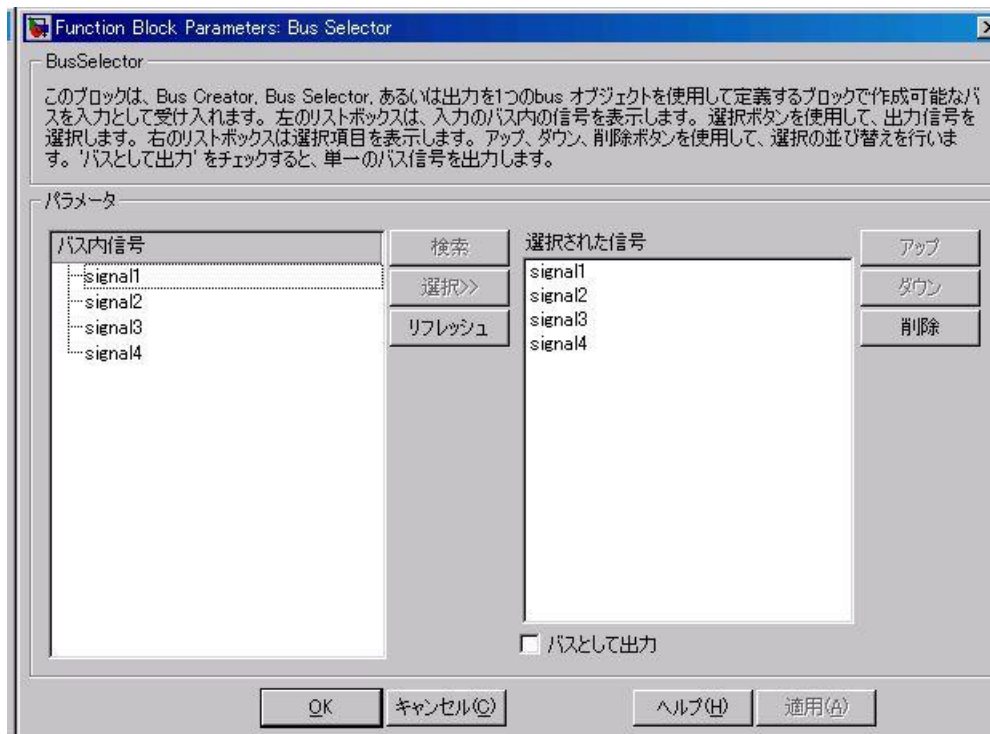
Bus Selector をドラッグ&ドロップして、Bus Creator と接続します。



Bus Creator の信号を選択します。



モデル名 : SIGNALBUS.mdl



Mux

目的 バスまたはベクトル信号の要素を多重化します。

ライブラリ **Signal Routing**



Demux

目的 バスまたはベクトル信号の要素を抽出して出力します。

ライブラリ **Signal Routing**



Demux ブロックは、入力信号の成分を抽出して、成分を個別の信号として出力します。ブロックはベクトル（1次元配列）信号またはバス信号を受け入れます。

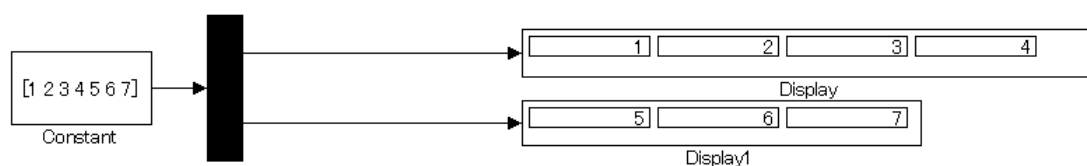
（詳しくは、Using Simulink マニュアルの“信号バス”を参照してください）

出力数 パラメータによって、各出力ポートの数と（オプションで）次元を指定することができます。出力の次元を指定しなかった場合、ブロックが出力の次元を自動的に決めます。

Demux ブロックは、バス選択モード パラメータが選択されているかどうかによって、ベクトル選択モードまたはバス選択モードで動作します。2つのモードは、受け入れる信号のタイプに違いがあります。ベクトルモードは、ベクトル状の信号、すなわち、スカラ（1要素配列）、ベクトル（1次元配列）、または列あるいは行ベクトル（1行または1列の2次元配列）だけを受け入れます。バス選択モードは、Mux ブロックまたは別のDemux ブロックの出力だけを受け入れます。

Demux ブロックの出力数 パラメータは、ブロックの動作モードに応じて、ブロック出力の数と次元を決めます。

モデル名 : demuxblock.mdl



あとがき

作成協力にあたって、本書の中で紹介されている JMAAB 活動の成果物（スタイルガイドライン・発表資料等）や論議の内容を引用して作成してあります。

まさに今、自動車の制御システム開発をしているエンジニア、マネージャの経験と知識・知恵が生かされている入門書にしましたので、開発をしている光景をイメージしながら学習してほしいと思います。

筆者等から相談を受け JMAAB を立ち上げ、その活動で育てられた私の知見が生かせれば幸いと思い協力させていただきました。この技術分野は、成長スピードが速く、日々新しい技術を学び創造することが多くありますので共に発展させていきましょう。

尾形 永（株式会社ミツバ）

※文書の引用について

本章の内容は、JMAAB からの引用があります。

記載された内容について JMAAB が保証するものではありません。また、書かれている内容に不具合が出た場合でも JMAAB では一切の責任を負いかねます。

参考文献

情報処理推進機構 SEC journal №13 自動車分野の MBD 技術者に必要なスキル(ETSS-JMAAB)
日経 Automotive Technology 2008 年 7 月号 p74

参考 URL

JMAABホームページ	http://jmaab.mathworks.jp/
MathWorks社ホームページ	http://www.mathworks.com/
情報処理推進機構(IPA)	https://sec.ipa.go.jp/

筆者

磯貝 孝夫	(株式会社両毛システムズ http://www.ryomo.co.jp/)
須永 充	(株式会社両毛システムズ http://www.ryomo.co.jp/)

モデルベース開発入門

MATLAB 及び Simulink は米国 The MathWorks 社の登録商標です。
その他の製品等の固有名詞は、それぞれ各社の商標又は登録商標です。
本書の内容の一部あるいは全部を無断で転載、複製、複写することを禁じます。
本書の内容は予告なく変更することがあります。

©2010 The MathWorks, Inc. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See <http://www.mathworks.com/trademarks> for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

平成 23 年度文部科学省委託
 東日本大震災からの復旧・復興を担う専門人材育成支援事業
 「東北の復興を担う自動車組込みエンジニア育成支援プロジェクト」

【委員名簿】

●推進協議会

- ◎佐藤 公一 東北電子専門学校 校長
 今野 幸信 東北電子専門学校 総務部長
 幸田 和明 花壇自動車大学校 校長
 與那嶺 尚弘 国立仙台高等専門学校 知能エレクトロニクス工学科 准教授
 伊藤 俊 宮城県黒川高等学校 教頭
 木村 康弘 宮城県米谷工業高等学校 情報技術科
 今野 基 宮城県工業高等学校 教頭
 渋谷 義博 トライポッドワークス株式会社
 技術本部 プロジェクトマネージメントグループ プロジェクトマネージャ
 白田 正樹 株式会社アペールジャパン
 林 毅 宮城県経済商工観光部 産業人材対策課 課長
 前田 晋佑 宮城県自動車産業振興室 自動車産業振興班 主事
 今井 和彦 宮城県産業技術総合センター 機械電子情報技術部情報技術開発班 副主任研究員
 吉岡 正勝 有限会社ザ・ライスマウンド マーケティングマネージャ

●開発分科会

- ◎坂藤 健 東北電子専門学校 自動車組込みシステム科 学科主任
 村岡 好久 名古屋工学院専門学校 テクノロジー学部 部長
 柴原 健次 エキスパート・プロモーション代表
 服部 博行 株式会社ヴィッツ 取締役 組込みソフトウェア開発部部長
 伊藤 政光 株式会社エスワイシステム 取締役オープンシステム部部長
 吉岡 正勝 有限会社ザ・ライスマウンド マーケティングマネージャ

●講座運営分科会

- ◎坂藤 健 東北電子専門学校 自動車組込みシステム科 学科主任
 小野寺 敬司 花壇自動車大学校 教頭
 與那嶺 尚弘 国立仙台高等専門学校 知能エレクトロニクス工学科 准教授
 村岡 好久 名古屋工学院専門学校 テクノロジー学部 部長
 伊藤 政光 株式会社エスワイシステム 取締役オープンシステム部部長
 吉岡 正勝 有限会社ザ・ライスマウンド マーケティングマネージャ

●事業実施協力専修学校・企業・団体等

- 古賀 稔邦 日本電子専門学校 校長
 石川 浩 日本工学院八王子専門学校 テクノロジーカレッジ ロボット科
 岡田 靖志 浜松情報専門学校 教務課長
 村岡 好久 名古屋工学院専門学校 テクノロジー学部 部長
 村上 登昭 大阪工業技術専門学校 教員
 羽曾部 恭美 カストマシステム株式会社 エンベデッドシステムソリューション事業部事業部長
 服部 博行 株式会社ヴィッツ 取締役 組込みソフトウェア開発部部長
 伊藤 政光 株式会社エスワイシステム 取締役オープンシステム部部長
 富田 茂 キャリア技研株式会社 代表取締役
 小林 靖英 株式会社アフレル 代表取締役社長
 柴原 健次 エキスパート・プロモーション代表
 吉岡 正勝 有限会社ザ・ライスマウンド マーケティングマネージャ
 飯塚 正成 一般社団法人全国専門学校情報教育協会 事務局長

◎＝委員長

平成 23 年度文部科学省委託
 東日本大震災からの復旧・復興を担う専門人材育成支援事業
東北の復興を担う自動車組込みエンジニア育成支援プロジェクト
モデルベース開発入門

平成 24 年 3 月
 東北の復興を担う自動車組込みエンジニア育成支援プロジェクト
推進協議会

連絡先：〒164-0003 中野区東中野 1-57-8
 (一般社団法人全国専門学校情報教育協会事務局)
 電話：03-5332-5081 FAX：03-5332-5083

●本書の内容を無断で転記、記載することは禁じます。