

# アノテーション自動表示による モデル可読性向上への取り組み

2014年12月18日

アイシン・エイ・ダブリュ株式会社

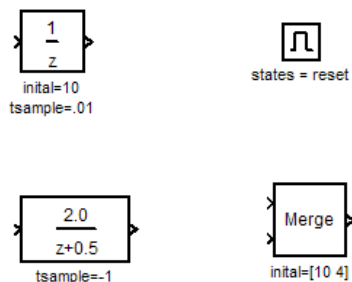
技術本部 第2制御技術部

主席研究員 久保孝行

- アノテーション表示が必要な背景
- アノテーション表示に使用する技術の紹介
  - SimulinkAPI
- 最初の実施内容の紹介
- 改善内容の紹介
- 応用事例
- まとめ
- その他(時間があれば)

- C言語同様、Simulinkを用いたモデル記述もスタイルガイドに準じたモデリングを行います。  
例えば、モデリングガイドラインにはMAAB定義のガイドラインがあります。

**db\_0140 : ブロックパラメータの表示**  
重要なブロックパラメータは表示されなければなりません。



可読性向上のため

**CONTROL ALGORITHM MODELING  
GUIDELINES USING MATLAB®, Simulink®,  
and Stateflow®**  
Version 4.0 (日本語版)

Japan MBD Automotive Advisory Board  
(JMAAB)  
2015年01月30日

© Copyright 2007 JMAAB. All rights reserved. 1

## 7.2. 各ルールの選択パラメータ

### 7.2.1. 解説

いくつかのルールは文書中に選択式である事が明示的に記載されていますが、その記述はすべてのルールに書かれている訳ではありません。それ以外についても、記載された通りにしなければならない訳ではありません。本ガイドラインは、あくまでもプロジェクトで活用する為に、ルールのテンプレートを提供しています。それぞれのガイドに書かれた数字やブロック種別は絶対的なものではなく、プロジェクトの特性に合わせて変化させる必要があります。ここではルールとしてプロジェクトの特性から決めなければならない最低限の選択肢を記載しました。そのほかにもプロジェクトのプロセスや制御対象の条件、携わるエンジニアの平均的なレベルから総合的に判断し、ガイドラインの真意を読み取り運用してください。

### 7.2.2. 一覧表

変更可能なパラメータ種類をすべて記載している訳ではありません。

ルールID	パラメータ
ar_0001	ファイル名の対応拡張子を決めます。 モデルファイルだけを対象にする場合は mdl と slx です。 MATLAB 系のファイルに限定する場合は下記が対象となります。 {m,p,mdl,slx,fig,c,hm 全てのファイルを対象
ar_0002	
jc_0241	総文字数
jc_0242	総文字数
jc_0201	
jc_0211	
jc_0222	
jc_0232	
jc_0231	対象のサブシステムの種 関数定義への拡張
jc_0240	総文字数

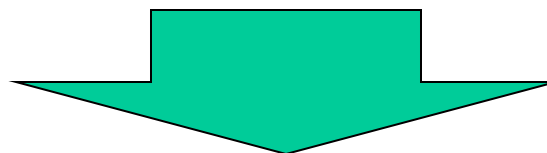
db\_0140

対象となるプロセス毎に、以下のリストを設定します。

- 対象ブロックタイプと表示すべきオプション名、表示する条件
- 表示方法、表示文字

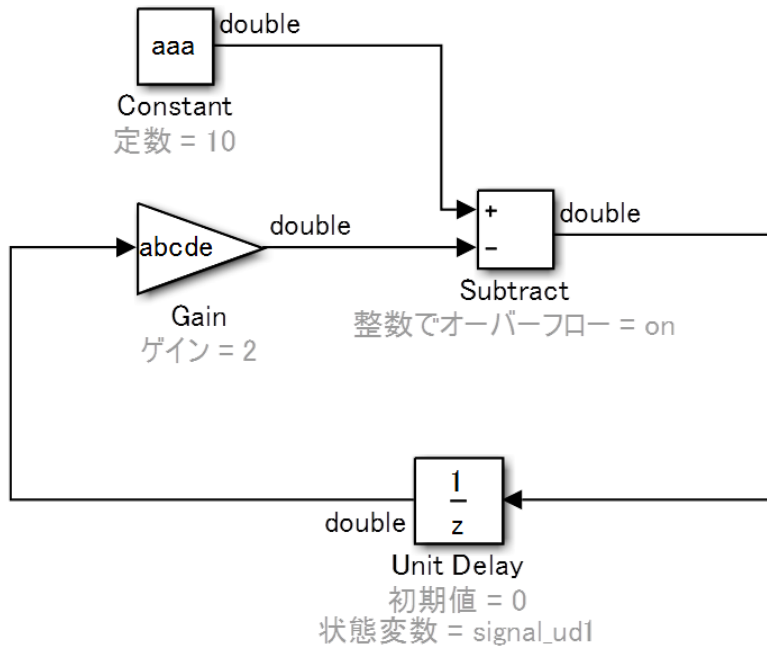
つまり、何を表示するかは、ユーザーが決める

- ブロック毎に何を表示しますか？
- どの様に表示しますか？
  - ブロックごとに表示したい内容が異なる。



内容を決定する

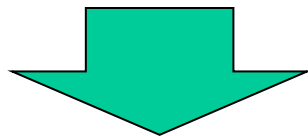
複数人から情報を集約  
17ブロックに対しての要求をまとめた



17ブロックについて表示すべき例題を作成し、  
ルールとして定義。

課題： アノテーション表示を人が実施する事が、  
工数的に、可能なのか？

- スタイルガイドへの準拠について、チェック機能を重視されがちですが、この例では、人がモデルに設定し、自動チェックを実施しても、ルール準拠の作業が膨大でルールが守られない可能性がある。つまりチェックしても大量にエラーを検出するだけで意味が無い。
- db\_0140: アノテーション表示は、シミュレーションやコード生成に影響が出ない。したがって、モデルを変更しても影響度が無いので、自動化する事が出来る。つまり、チェックではなく、必要な時にボタンを押すようにして対応する事が可能。



**自動表示の仕組みを開発する**

## Simulink APIを使います。

### API【Application Program Interface】

- Simulink API とは  
MATLABのコマンド操作にて、Simulinkの外部からSimulinkモデルを操作する事が可能なインターフェース関数



Simulinkのヘルプに記載されています。



MATLABのコマンドラインから

>> Simulink

Simulinkライブラウザの起動

>> open\_system('モデル名.mdl')

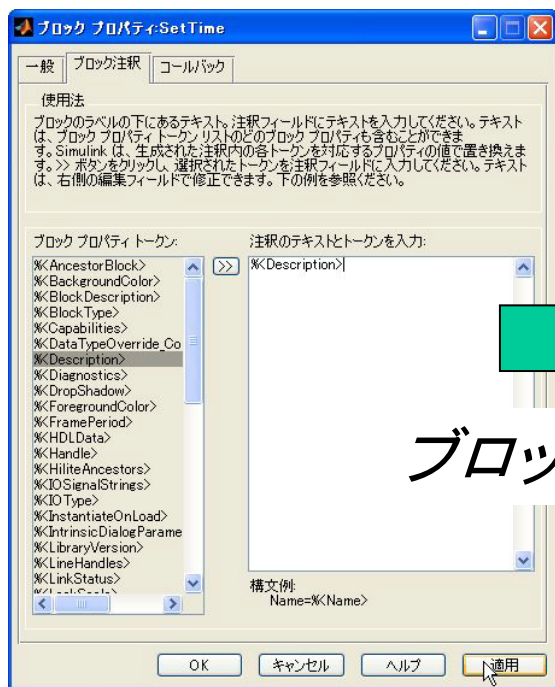
モデルのオープン

これもSimulink APIです。

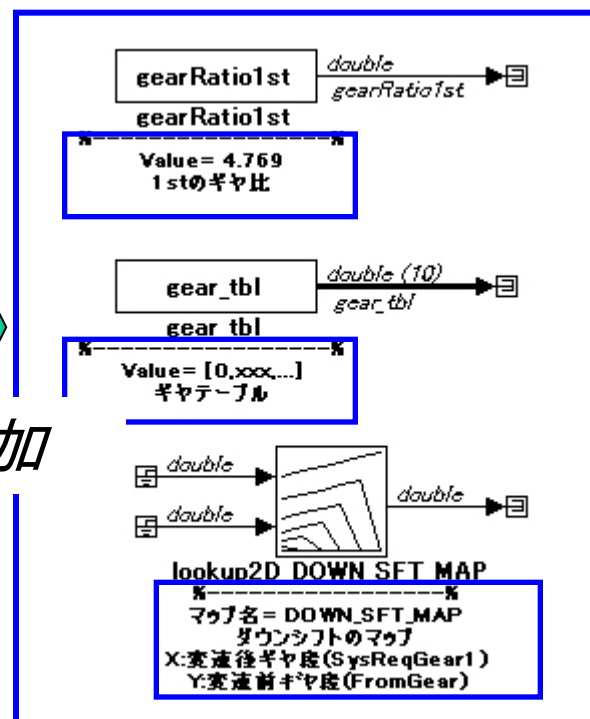
コマンドラインからSimulinkのモデルを操作できます。

要望: ブロックの概要を表示したい。

- 人が実施する場合、マウス操作によりブロックプロパティ画面を開き、ブロック注釈に、“概要”を表示するように選択する。
- モデル内に数十個ブロックがあった場合、作業は一瞬では終わりません。

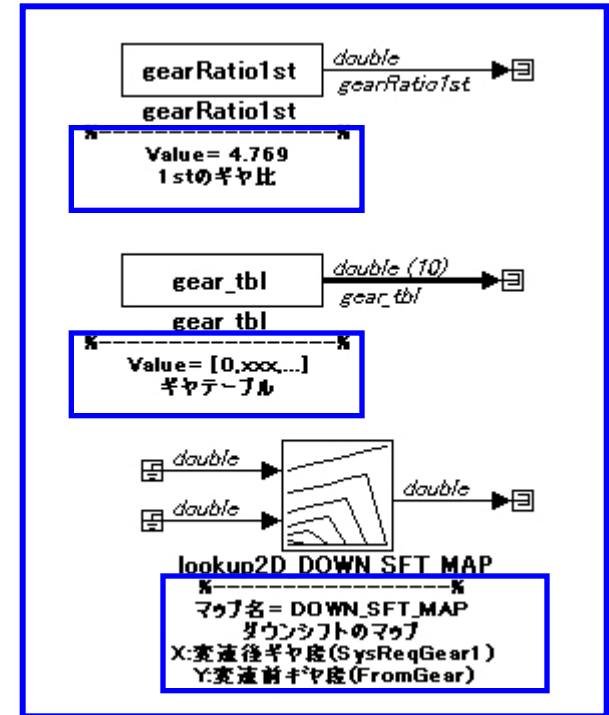


ブロック注釈の追加



- Simulink APIを使う場合、  
わずか数行のプログラムを書いて、  
実行すれば、全てのブロックに対して  
数秒で作業が終了する。

Simulink APIは非常に便利



(MATLABコマンドラインに切り取って実行する)

```

modelH=get_param(bdroot,'Handle');
CblockH=find_system(modelH,'LookUnderMasks','all','type','block');
for n1=1:length(CblockH)
    set_param(CblockH(n1),'AttributesFormatString','%<Description>');
end
    
```

- ブロックのタイプ毎に、何をどのように表示させるか設計し、実行させる。

## プログラム例

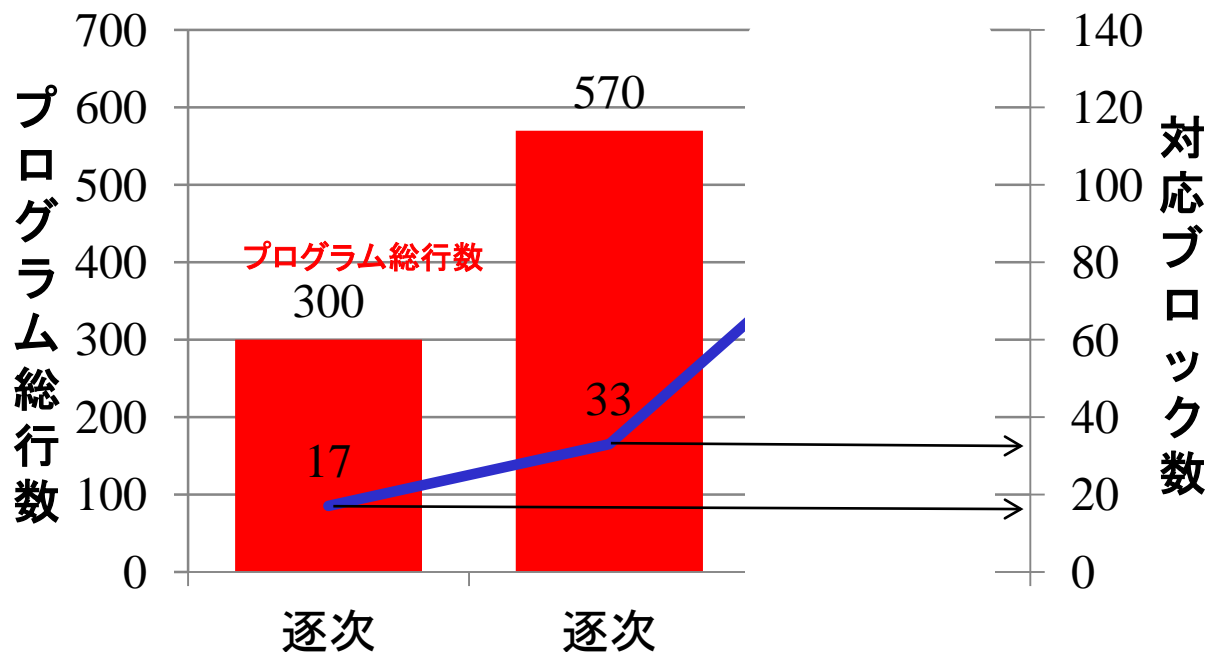
```
for blkNo=1:length(blockList)
    if strcmp(blockType,'BusCreator')
        .....
    elseif strcmp(blockType,'Constant')
        .....
    elseif strcmp(blockType,'Gain')
        .....
```

## 欠点

- ブロックが追加されるたびに、行数が増える。
- 見たいオプションが変わると、プログラムを修正する。

手直しの工数が莫大で、実運用できない。

徐々に対応ブロック数が増え、17ブロックから33ブロックになった時に、コードサイズがほぼ倍増



対応ブロックの種類やオプションの追加ニーズは次々に出てくる。

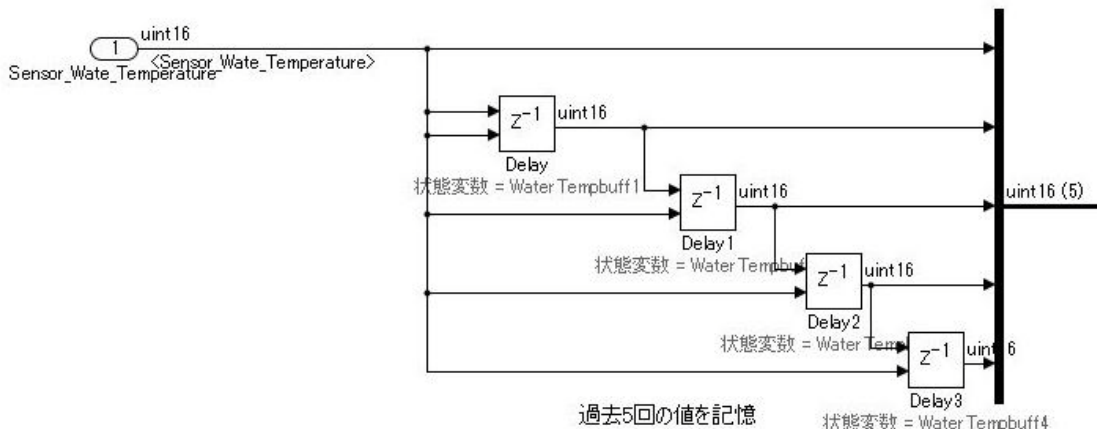
ここで、もう一度ニーズを集約、整理した。

改善に向けて

1. どの様に表示しますか。
  - ブロック毎に何を表示しますか？
  - どの様に表示しますか？
    - ブロックごとに表示したい内容が異なる。
2. 誰が重要と判断しますか？
  - 機能を検討する人
  - 実装用にコード生成の設定を行う人
  - 検査を行う人

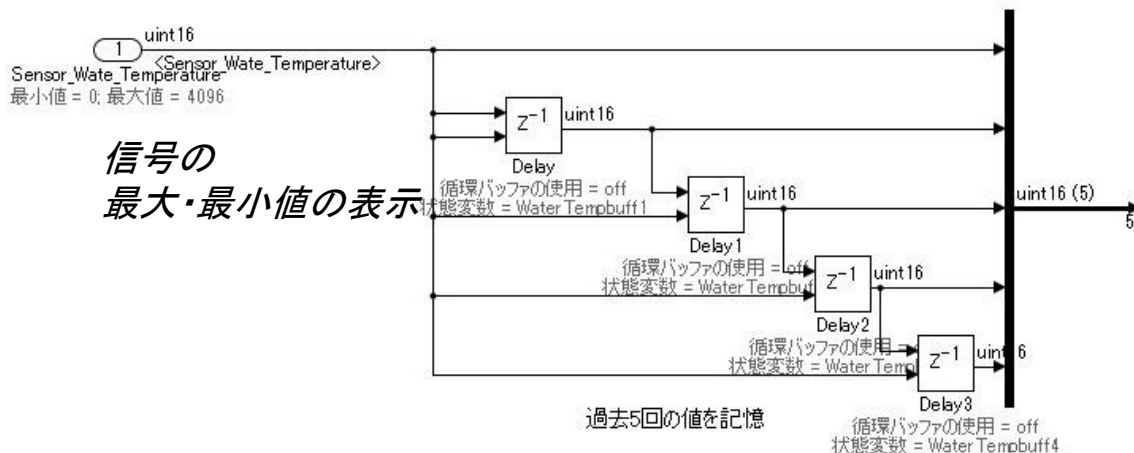
プロセス担当者によって、表示したい内容が異なる

機能を考えるユーザーは、最小限のオプション表示が良い。



表示が多すぎると  
仕様の理解の邪魔

実装設定を行うユーザーは、実装に影響する情報が欲しい。



コードに影響するオプションを表示

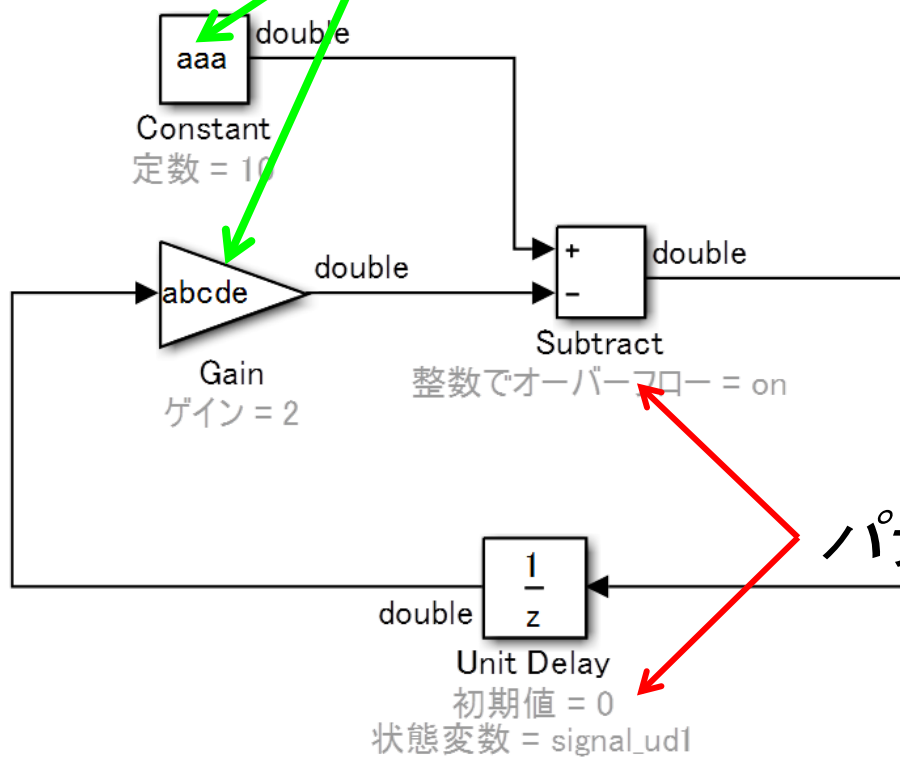
整理すると3職種:2種類に集約可能



アノテーション表示のオプション毎の操作内容を調査、分類した

1. 個々のブロック固有のオプションを用いた表示
  - UnitDelayはinit
  - ConstblkブロックはValue
2. 共通するオプションを用いた表示
  - サンプリング周期
  - プロパティ
3. 状況によって、アクティブになるオプションを用いた表示
  - Enableサブシステム内に存在するoutportブロック

見ただ目でパラメータが解る



パラメータが隠れている

## 見せ方の種類を5種類に統合

- 表示パターンを指定する。
- ブロック毎に表示オプションを指定する。
- 特殊ケースは、関数名を記載する。

これらの対応で、プログラムの統合とプログラムの一部をデータへ移動可能！

a.blocktype={'Constant'};	ブロック種類
a.Option={'Value'};	表示オプション
a.OptionType={'1-A'};	見せ方
a.SpecialCond = [];	
a.text.eng={'Value'};	表示文字(英語)
a.text.jp={'定数'};	表示文字(日本語)

```
a.blocktype={'Constant'};
a.Option={'Value'};
a.OptionType={'1-A'};
a.SpecialCond = [];
a.text.eng={'Value'};
a.text.jp={'定数'};
a.text.disp=[];
ShowAnnotationParameter(end+1)=a;
a.blocktype={'UnitDelay'};
a.Option={'X0','StateIdentifier'};
a.OptionType={'1-C','2'};
a.SpecialCond = [];
a.text.eng={'Ini Value','StateIdentifier'};
a.text.jp={'初期値','状態変数'};
a.text.disp=[];
```

ブロック毎に構造体に  
オプションを指定する。

対応ブロックが増えると  
構造体を追加していく。

関数を呼び出す宣言ができる。

```
a.blocktype={'Switch'};
```

```
a.Option={'Criteria','Threshold'};
```

*@sp\_func\_Switch:関数ハンドルを使用する。*

```
a.OptionType={'2','3'};
```

```
a.SpecialCond = @sp_func_Switch;
```

```
a.text.eng={'PassingCondition','Threshold'};
```

```
a.text.jp={'通過条件','閾値'};
```

```
a.text.disp=[];
```

```
ShowAnnotationParameter(end+1)=a;
```

%% Switch固有の特殊処理

```
function spstr = sp_func_Switch(varargin)
```

```
    spstr = "";
```

```
    Criteria = get_param(varargin{1}, 'Criteria');
```

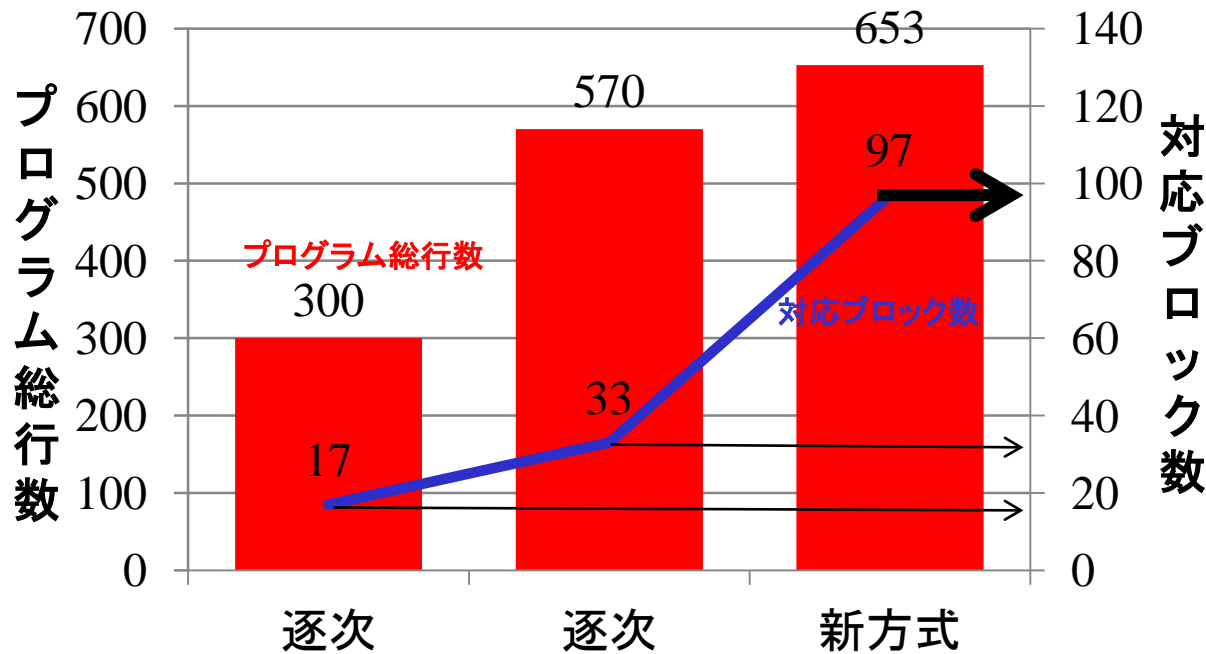
```
    if ~strcmp(Criteria, 'u2 ~= 0') % 条件式が'u2 ~= 0'のとき、通過条件、閾値を表示する。
```

```
        spstr = [varargin{3}, ' = ', '%<',varargin{2}, '>'];
```




```
    end
```




```
end
```

対応ブロック数33と新方式の97ブロック対応で、プログラム総行数にはそれほどの違いがない。対応ブロック数が少なければ、個別の対応が望ましく。およそ30ブロックを超える対応が必要な場合に本手法が有効である事が解る。



- ユーザーの職種毎にデータを切り替える事で、必要とされる表示パターンを変更できるようになった。
  - 対象ユーザーの複数化に対応

 Code\_data\_ALL\_annotationparameters.m  
 Code\_data\_Block\_annotationparameters.m  
 Code\_data\_Mask\_annotationparameters.m

 Contrl\_data\_ALL\_annotationparameters.m  
 Contrl\_data\_Block\_annotationparameters.m  
 Contrl\_data\_Mask\_annotationparameters.m

- MATLABは、テキストデータを、そのまま実行できる。
- 追加したプログラム(データ)はコンパイルが不要。

この機能を活用する事で、プログラムを変更しなければならないようなユーザーニーズに柔軟に対応できる環境を構築する事ができる。

- 応用: 複数の流派があるガイドライン
  - ブロックサイズ
  - ブロックの色の使い方
  - .....



- 同様の仕組みを利用すれば、ブロックサイズの調整も、プロジェクト、あるいはチーム単位で標準化した設定にできる。

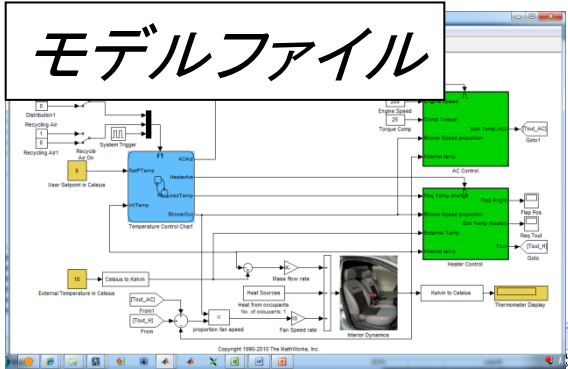
## データの設定例

データに文字を含め  
可変長の数式に対応

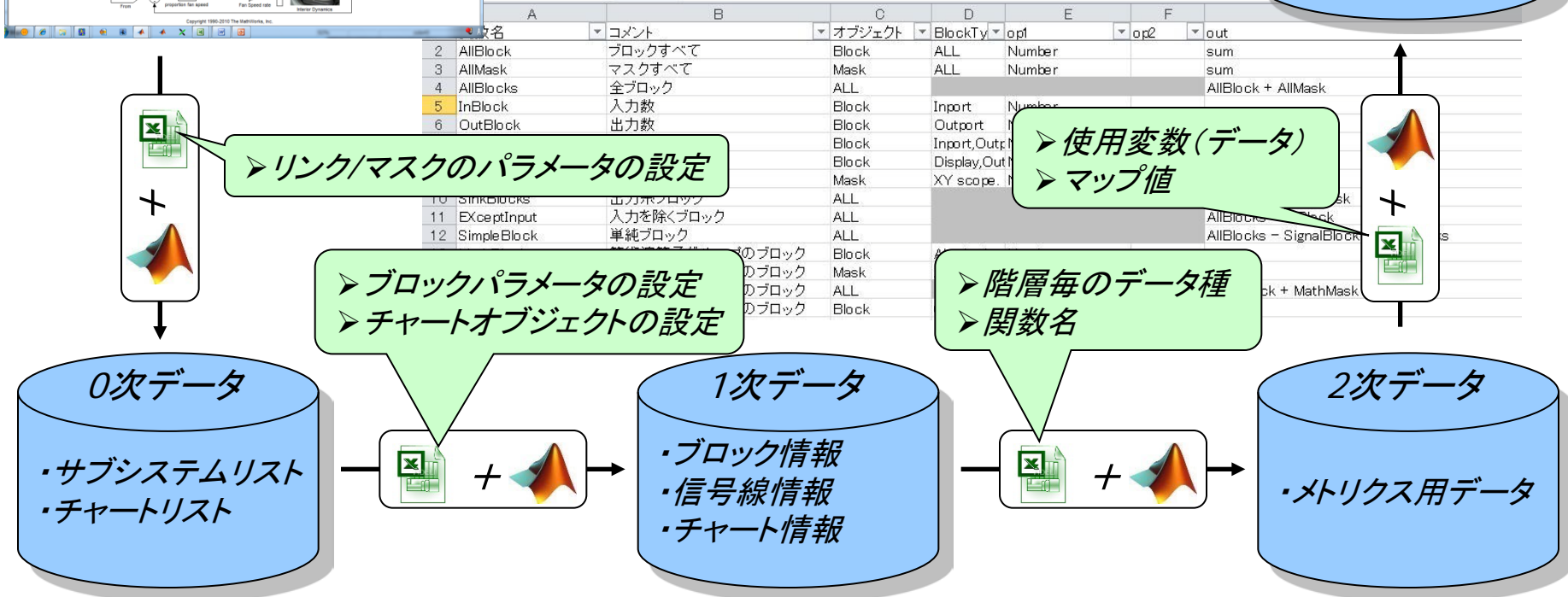
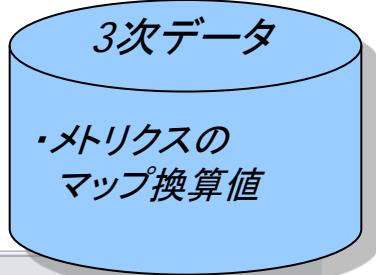
BlockType	MaskType	option1	Comparison Name	option2	width	hight	X_VariableSize	Y_VariableSize	MaxWidth	MaxHight
SubSystem				SubsystemLength	100	42	num*7	numIO*30	500	1000
Masked	Default			SubsystemLength	30	30	numIO*7	numIO*50	500	500
Masked	Stateflow			SubsystemLength	30	30	num*7	numIO*50	500	500
Bitwise Operator					60	40				
Data Type Propagation					40	-1				
Delay		InitialConditionSource	Input port		55	40				
Delay					40	40				
Function-Call Generator		numberOfIterations								
Function-Call Generator										
Integer Delay										
Lookup Table Dynamic					55	50				
Tapped Delay Line					40	40				
Abs					30	30				
Bias					40	-1				
Bit Clear					60	40				
Bit Set					60	40				
Block Support Table					40	38				
BusAssignment					-1	-1				
BusCreator					5	38		(19 * numI)		500
BusSelector					5	38		(19 * numO)		500

特殊な関数の呼び出しで  
複雑な処理を実行させる

この例では、mファイルの構造体形式をやめて、  
エクセルファイルで管理できるようにした。



エクセルファイルを使って  
モデルが持つ情報の  
数を測定するツールを作成



ユーザーは、MATLAB言語を知らなくても、  
モデルから必要な情報の数を取得できる

- アノテーション表示ツールを作る事で
  - 見やすさが改善され、レビュー時間の削減が実現できた。
  - レビュー時の視点によって表示を切り替える事ができる。
  - db\_0140については、ガイドラインチェッカーを使用する必要がなくなり、検査工数の手間が削減できた。
- プログラムを一部データ化する事で
  - メンテナンス性が向上し、ユーザーがチーム単位でメンテナンスできるツールを作る事が出来た。
  - 同様の手法を他の機能に展開する事で、複数個の機能について、メンテナンス性が向上した。

波形形式A  
適合ツール

波形形式B  
HILS:結果

入力



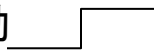
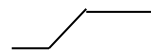
波形形式D  
単体検査

波形形式E  
自動検査

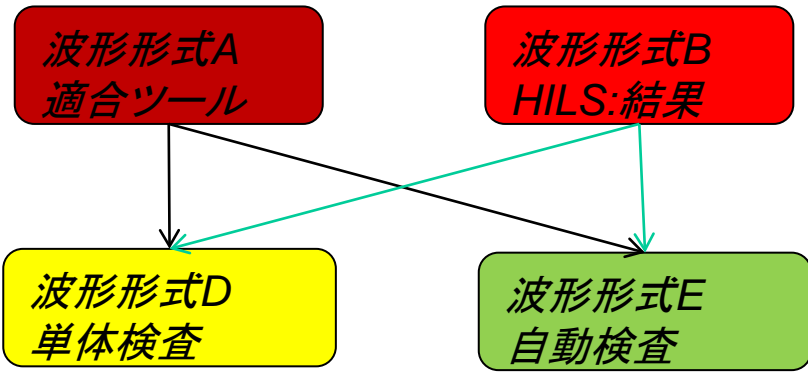
出力

作り方  
1. 全ての組み合わせを作る  
A→D,E  
B→D,E

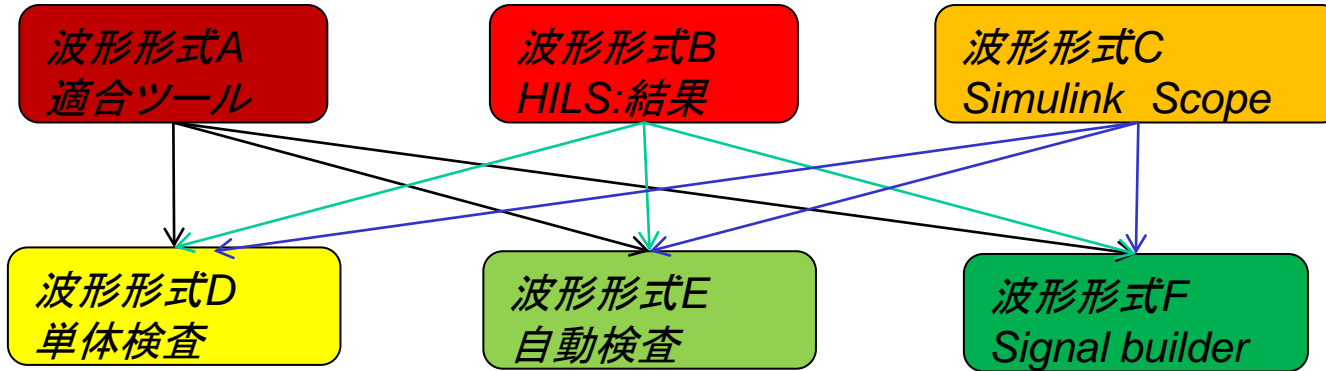
波形に含まれる様々な情報と形式

- 時間一つに複数の信号データ
- 時間と信号データのセット
- 同一時間のステップ移動 
- 信号ヘッダ情報
- 軸情報 

## 4通りの変換を作ればよい



## それぞれのフォーマットに対して、個別に変換プログラムを作る



作り方

1. 全ての組み合わせを作る

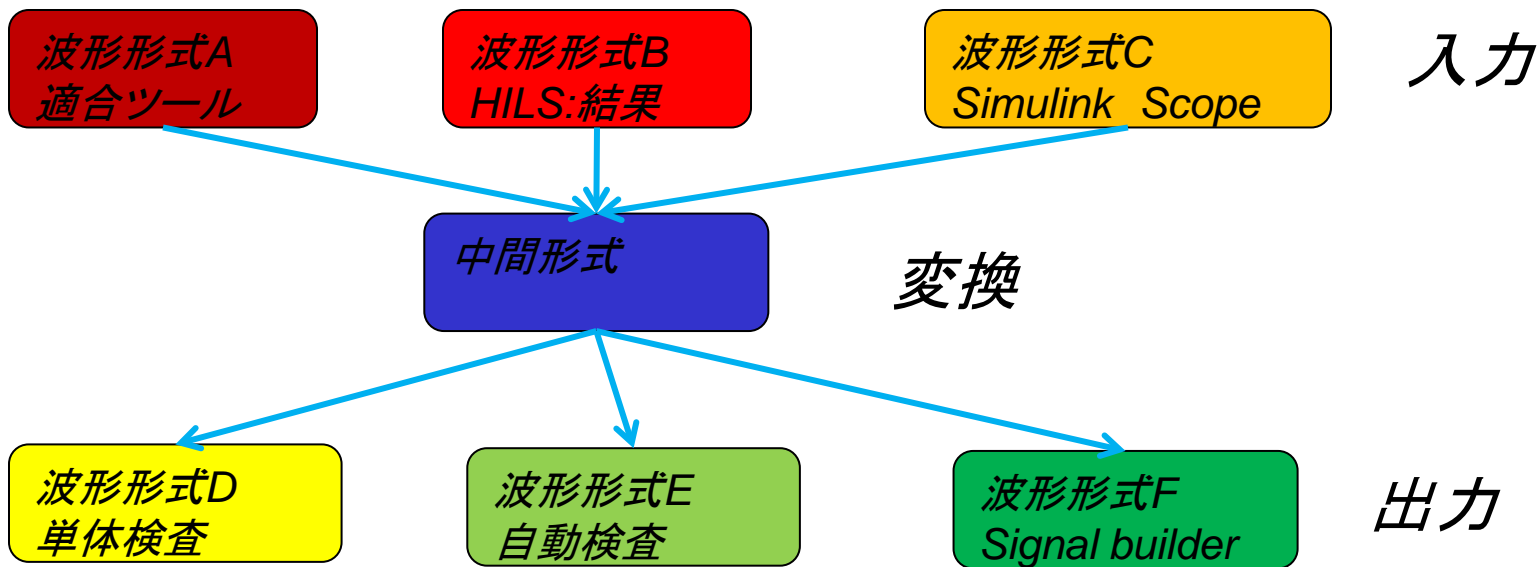
A→D,E,F

B→D,E,F

C→D,E,F

種類が増えると、組み合わせが増えて、  
すぐにプログラムが複雑化。  
メンテナンス不可能となる。

9通りの変換を作る。



作り方

2. 共通フォーマットを作る

A→G

G→D

B→G

G→E

C→G

G→F

共通のデータフォーマットを作り、  
片側の変換を用意する。  
この場合は、6通りの変換を作れば良い

このような設計手法は、制御系組みソフトウェア開発とは異なるので、慣れが必要。  
従来のソフトウェアエンジニアから開発環境エンジニアに職種切り替えを実施する場合、  
単にプログラム言語がMATLABに切り替わるだけでなく、  
プログラムテクニックが大きく異なります。  
これらの事を習得するには、訓練が必要です。

- MATLAB言語を用いた環境開発を受託してくれる会社はありますが、メンテナンス性まで考慮してツールを作ってくれるところはほぼありません。

## 「なぜか！」

- 単発のプログラム開発は早い方が好まれる。
  - 安かろう、悪かろう。ユーザーが同じ事が出来るなら安い方を好む。
- 保守、メンテナンスを受託する場合、工数が多い方が受託会社が儲かる。

## 騙されない対策は。

- 発注側にそれを見極める能力が無ければ、騙されたままになるわけです。発注側も、技術的なテクニックを身に付け、嘘偽りを見抜く事が出来なければいけません。