Vol.2016-SE-191 No.30
2016/3/15

制御ソフトウェア向け機能テストケース生成方式の提案

磯田　誠†　西山博仁†

概要：制御装置に搭載される制御 S/W は，多機能化・付加価値向上に応えるための制御機能の電子化により，大規模化・複雑化が急速に進んでいる．さらに今後，派生機種や仕向け地の違いによる制御 S/W のバリエーションの増加により，細分化が進んでいくことが見込まれる．このような特徴を持つ制御 S/W に対して従来の開発プロセスを適用した場合，特に試験工程でのリソース面の問題から開発コストが上昇しており，解決策として試験作業を効率化する機能テストケース生成方式を提案する．

具体的には，試験工程に対する要求事項を機能に基づく網羅と構造に基づく網羅の同時保証と規定し，これを実現するための各種試験手法の適用方法および機能テストケースの自動生成方法を考案する．また，実開発に導入した際に試験作業を妨げないことを考慮して，自動生成にかかる時間を短縮する方法も盛り込む．

題材とする制御 S/W への適用例と評価結果から，施策実現により得られると想定した「十分な試験項目一式を決定できる」という効果が見込めると判断する．

今回の適用例では各種試験手法の中から最も基本的な同値分割・境界値分析と分岐網羅を取り上げた．検証手法としては各種試験手法に容易に拡張できるようにしており，今後は各種試験手法に適用範囲を拡大して評価を進め，本方式の有用性を向上させていく予定である．

キーワード：製品品質，機能網羅，構造網羅，有界モデル検査，モデルベース開発

An Approach of Functional Test Case Generation for Control Software

MAKOTO ISODA†　HIROHITO NISHIYAMA†

Keywords: Product Quality, Functional Coverage, Structural Coverage, Bounded Model Checking, Model-Based Development

1. はじめに

制御装置に搭載される制御 S/W は，多機能化・付加価値向上に応えるための制御機能の電子化により，大規模化・複雑化が急速に進んでいる．さらに今後，派生機種や仕向け地の違いによる制御 S/W のバリエーションの増加により，細分化が進んでいくことが見込まれる．このような状況下で収益力を維持・強化するには，制御 S/W 開発の生産性向上に取り組む必要がある．

一方，制御装置を一から新規開発することは稀で，既存の装置を流用して機能変更することがほとんどである．このような特徴を持つ制御 S/W についても製品品質確保と省リソースの両立がニーズとなることに変わりはない．しかし，これに対して従来の開発プロセスを適用した場合，特に試験工程でのリソース面の問題が顕在化しており，開発コストを押し上げている．

そこで本稿では，上記問題の主要因となっている試験作業量の爆発傾向に対して，開発の全作業量をむしろ圧縮することを目標に，試験作業を効率化する機能テストケース生成方式を提案する．また，制御 S/W の題材を用いて，本方式の適用例と評価結果を示す．

2. 製品品質とリソースの両立が求められる制御ソフトウェア開発

2.1 開発プロセス上の課題と要求事項

既存の装置を流用して機能変更する場合は，一から新規開発する場合に比べて難易度が向上するため，ニーズを満たすには複数機種にまたがる再利用に焦点を当てた製品系列開発[1]の実現がキーになると考える．製品系列開発とは，複数機種間でアーキテクチャを共通化し，機種間の共通部品と機種ごとの固有部品を用いて個々のシステムを開発するアプローチである．このアプローチのプロセス要件[2][3]と実開発の状況を照らし合わせ，試験工程の主要な問題点を図 1 のように想定する．

- 機能変更に対する S/W 試験が十分かどうかの判断が難しく，システム試験に作業量を先送りしている．

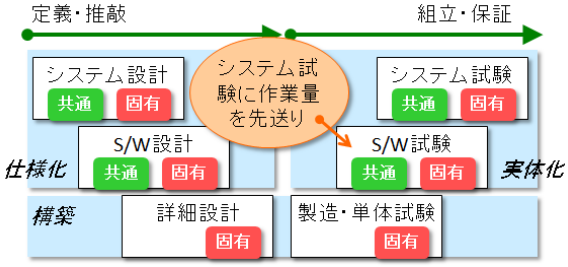

図 1　試験工程の主要な問題点

Figure 1　Major problem on software testing phase

† 三菱電機 (株) 情報技術総合研究所
Mitsubishi Electric Corporation Information Technology R&D Center

上記の問題の解決にあたって，試験工程に対する要求事項を規定する必要がある．そこで，人命の危険に直結するため高い安全性が求められる製品を想定し，機能安全規格（例えば航空分野の DO-178B[4]や自動車分野の ISO26262[5]など）を明確化した要求事項[6]を採用する．

(1) 制御 S/W の外部的な機能仕様に基づく網羅（以降，機能網羅）の保証．

(2) (1)に加えて，制御 S/W の内部的な構造に基づく網羅（以降，構造網羅）も同時に保証．

もしこれらの要求事項を個別に実現するのであれば，(1)については要求ベーステスト，同値分割，境界値分析といった機能に基づく試験手法，(2)については分岐網羅，MC/DC（Modified Condition/Decision Coverage）といった構造に基づく試験手法を用いることができる．

2.2 課題解決のための施策と実現方法

2.1 に示した個別の試験手法では，機能網羅に加えて構造網羅を同時に保証することができない．そのため，構造網羅を保証できなかった部分を人手作業で補わなければならないという欠点があり，作業もれを防ぐことができない．

これに対して我々は，機能に基づく試験手法と構造に基づく試験手法を適切に組み合わせて機能網羅と構造網羅を同時に保証する施策を立案する．これにより，十分な機能試験項目一式を決定することを狙う．課題と施策の位置づけを図 2 に示す．本施策の立案に当たり，開発基盤技術の一つとして，Simulink（MathWorks 社）を用いて制御 S/W を設計・実装・検証するモデルベース開発を採用した．

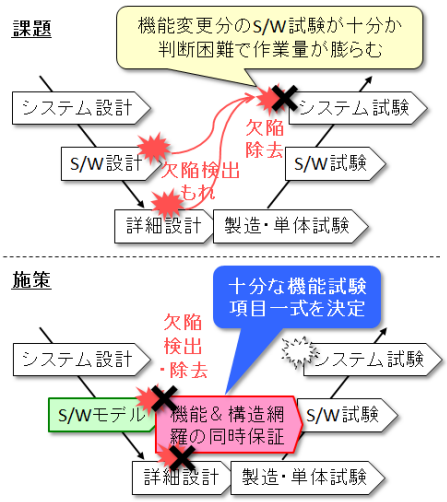

図 2 課題と施策の位置づけ

Figure 2 Focused problem and our solution

図 2 に示した施策の具体的な実現方法を以下に示す．

(1) 制御 S/W の外部的な機能仕様で指定される入出力条件を解析することで，機能テストケースすなわち時間軸上の入出力信号列を自動生成する．

(2) (1)の機能テストケースを自動生成するときに，制御 S/W の内部的な構造を網羅するための条件も同時に指定して解析する．

2.3 施策実現で得られる想定効果

本施策を実現した際に得られる想定効果を図 3 に示す．図 3 は，プロジェクトで承認され構成管理されている S/W の特定バージョンを表すベースラインを母体として，制御 S/W を開発する状況を想定している．図 3 では，開発対象 S/W を以下の 3 つにさらに細分化している[6]．

(a) 意図した要求事項の実現
母体に対する新規部分，変更部分，変更なし部分の合計．この部分はプロセスを踏んで開発していく．

(b) 想定外の要求事項の混入
母体に対する削除部分であり，本来は存在しない．しかし，既存 S/W の流用過程で削除困難や削除忘れのため残ることがある．試験もれが起こりやすく危険．

(c) どんな条件でも実行不能
論理矛盾のため決して実行できない部分．実行不能である確実な根拠を示し，無害を証明する必要がある．

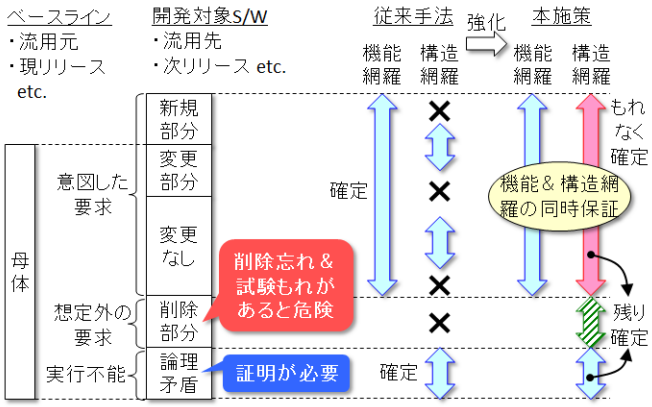

図 3 施策実現で得られる想定効果

Figure 3 Expected performance of our solution

試験作業量の爆発傾向は，人手による作業もれ（×印）をなくすための多大な労力によるところが大きい．本施策では，意図した要求事項の実現については機能網羅と構造網羅を同時保証することで特定し，どんな条件でも実行不能については従来手法で特定し，いずれでもない部分を想定外の要求事項の混入と特定することで，開発対象 S/W に対する十分な機能試験項目一式を決定できるようになる．

3. 機能網羅と構造網羅を同時保証する機能テストケース生成方式

3.1 検証対象とする S/W アーキテクチャ

本方式で対象とする S/W アーキテクチャを図 4 に示す．

- 機能要求を実現するアプリケーション，機能動作の仕組みである実行環境（通信，スケジューラなど），ドライバ・I/O デバイスのレイヤで構成される．
- アプリケーションは周期的に指定順序で起動される複数の制御処理で構成される．
- 実行環境とドライバは，レジスタアクセスなどの I/O 処理と，制御処理よりも優先度が高い割込みハンドラ

とタイマハンドラで構成される．

- アプリケーションと実行環境・ドライバの間でデータをやり取りしながら，時々刻々と制御演算を進める．

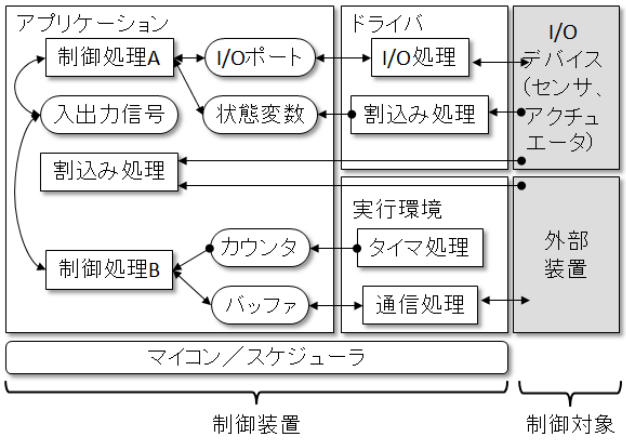

図 4　対象とする S/W アーキテクチャ

Figure 4　Assumed software architecture

3.2 検証カバレッジおよび検証項目の定義

本方式による品質保証範囲すなわち検証カバレッジを，図 4 に示した S/W アーキテクチャで動作する制御 S/W のアプリケーションの機能的な欠陥とし，これを検出するための検証項目を表 1 に定義する．

表 1　制御 S/W の検証項目

Table 1　Verification cases for control software

分類	検証項目	検証内容
機能要求の実現	期待通りの動作	特定の入力を与えたときに，要求として期待する出力となること．
	要求の実現度合い	構造網羅を基準に，意図した要求事項が十分に実現されること．

3.3 検証手法

3.3.1 前提と手法概要

検証対象とする S/W に対する前提を以下に示す．

- アプリケーション，実行環境，ドライバ・I/O デバイスのレイヤが分離されていると想定．分離されていない場合は，仕様書・コードを分析して予め分離する．
- アプリケーション部分を PC 上で動作させるため，ドライバを検証用スタブに，I/O デバイス／外部装置を制御対象モデルに置き換える．
- 実行環境もドライバと同じ方法で置き換える．ただし，H/W アクセスしている部分だけを置き換えるかまるごと置き換えるかは，難易度で判断．

上記の前提の下で，表 1 に示した検証項目に関する検証手法の概要を図 5 に示す．

- 信号ブロックとコードをリンクしたものを検証対象とし，これに検証用スタブと制御対象モデルを接続．
- 予め入力信号と期待する出力信号を作成しておき，実際の出力信号と比較して合否判定．

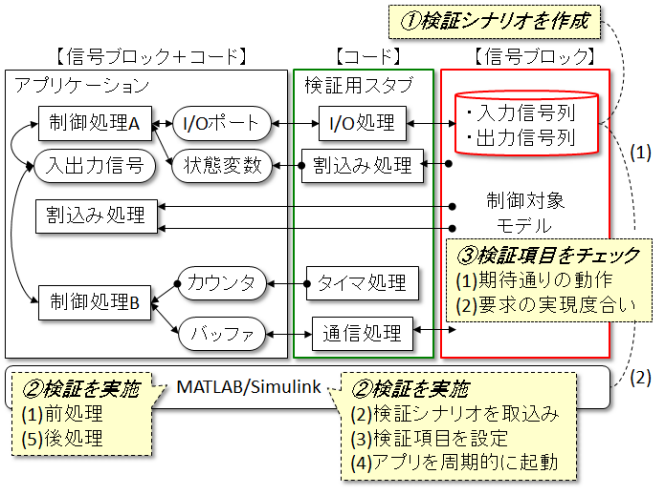

図 5　検証手法の概要

Figure 5　Overview of verification method

3.3.2 検証シナリオの作成方法

表 1 に示した検証項目に対して，検証シナリオの作成方法を表 2 に示す．これは図 5 の中の①に対応する．本稿では，機能テストケース一式のことを検証シナリオと呼ぶ．

表 2　検証シナリオの作成方法

Table 2　Methods to make verification scenario

分類	検証項目	作成方法
機能要求の実現	期待通りの動作	既存の試験手法を利用してモデルに機能仕様の入出力条件を指定して解析することで，入力信号列と期待する出力信号列を自動生成．
	要求の実現度合い	上記の自動生成において，モデルおよびコードの構造を網羅する条件も同時に指定して内訳を解析．

表 2 に示した検証シナリオの作成方法を自動化する本方式のフローを図 6 に示す．

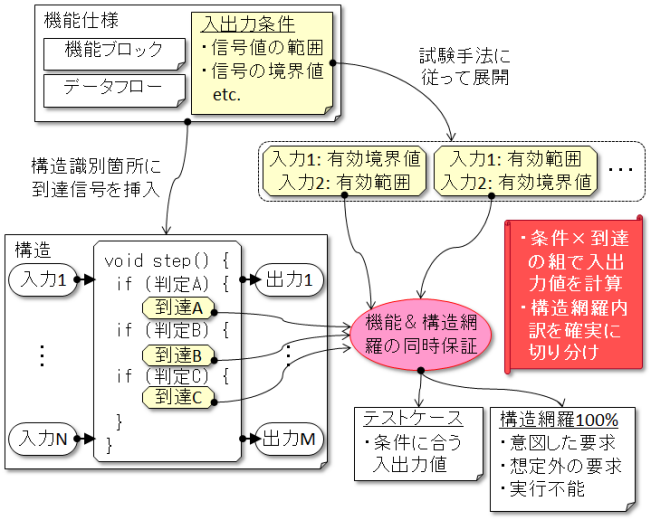

図 6　機能テストケース生成の自動化フロー

Figure 6　Automated flow for functional test case generation

検証シナリオの作成に応用する代表的な試験手法の入出力条件・選択基準を表 3 に示す.

表 3　代表的な試験手法の入出力条件・選択基準

Table 3　Input/output conditions and selection criteria

試験手法	入出力条件	選択基準 (※)
同値分割	信号値の範囲	全ての入出力信号が有効範囲
		ただ一つの入出力信号が無効範囲
境界値分析	境界値とその前後	ただ一つの入出力信号が有効境界値
		ただ一つの入出力信号が無効境界値
原因結果グラフ	条件間依存関係	境界値分析と同様 (入出力条件が増える)
代表値組合せ	信号の境界値，中間値他	入出力信号の代表値の単純羅列，総当たり他
Back-to-Back テスト	上記の各手法と同様	検証対象モデルとベースラインに対して同一の入力信号で出力信号が一致
分岐網羅	Don't care	実行不能を除くすべての分岐を実行する任意の値

(※) 有効（無効）境界値とは，境界値とその前後の値の中で，有効（無効）範囲に入る値.

表 3 に示した試験手法の入出力条件の指定方法を表 4 に示す.

表 4　入出力条件の指定方法

Table 4　How to specify input/output conditions

入出力条件	説明
ビット	入出力信号が取りうるビットパターンの有効値／無効値.
論理値	入出力信号が取りうる真偽値.
値の列挙	入出力信号が取りうる離散的な有効値／無効値.
値の範囲	入出力信号が取りうる連続的な有効値／無効値.
上記条件の組合せ	ビット，論理値，値の列挙，値の範囲を論理演算子で結合.

表 3 と表 4 に示した入出力条件は，図 6 に示した自動化フローの中の機能仕様の一部として指定し，これを用いて試験手法に従って具体的な条件式に展開する．検証対象の機能仕様の一部として入出力条件を指定した例を図 7 に示す.

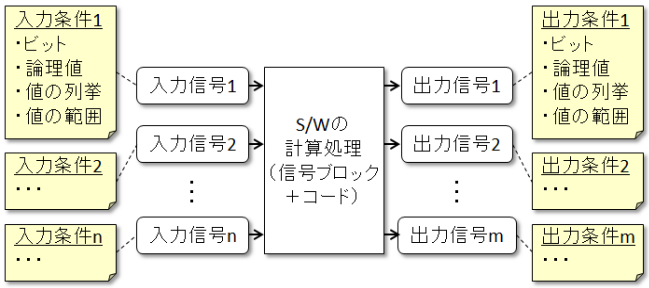

図 7　検証対象に入出力条件を指定した例

Figure 7　Input/output conditions example

また，図 6 に示した自動化フローの中で，検証対象の構造識別箇所に到達信号を挿入した例を図 8 に示す.

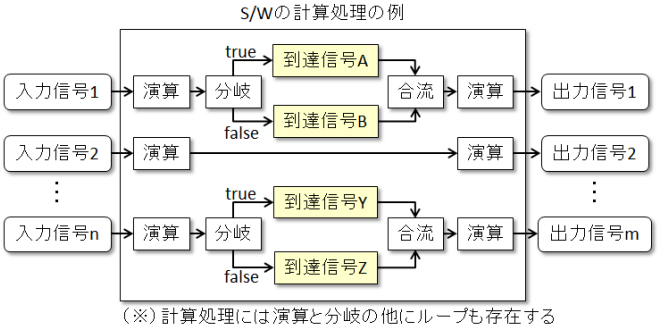

図 8　検証対象に到達信号を挿入した例

Figure 8　Instrumented reachability signals example

また，図 6 に示した自動化フローの中で，機能網羅と構造網羅の同時保証の全体集合を，入出力条件と構造識別箇所の総当たりと定義する．この全体集合の要素の一例として，図 7 に示した入力条件 1・出力条件 1 および図 8 に示した到達信号 A を組み合わせたものを図 9 に示す.

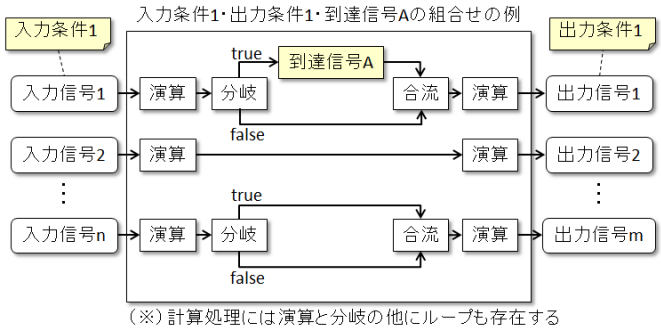

図 9　入出力条件と到達信号を組み合わせた例

Figure 9　Input/output condition and reachability signal example

以上で検証シナリオを作成するための情報が揃うが，入出力条件と構造識別箇所の総当たりすべてについて機能テストケースを生成しようとすると，組合せ爆発を起こす可能性が高い．これを回避する方法は 3.3.3 で述べる.

3.3.3 検証項目のチェック方法

検証項目のチェック方法を表 5 に示す. これは図 5 の中の③に対応する.

表 5　検証項目のチェック方法

Table 5　Methods to check verification case

分類	検証項目	チェック方法
機能要求の実現	期待通りの動作	・特定の入力を与えたときの実際の出力が，要求として期待する出力と一致すれば成功と判定. ・出力が異なる場合は，機能変更部分は人手で要確認，実装変更部分は失敗と判定.
	要求の実現度合い	構造網羅の計測結果から判断[6]. ・実行した箇所は，意図した要求事項の実現と自動判定. ・未実行箇所は，想定外の要求事項の混入と自動判定. ・どんな条件でも実行不能な箇所は，従来手法で自動特定.

検証項目のチェックに対する試験手法の適用方法を図 10 に示す．図 10 は適用する試験手法の間の依存関係を表すものであり，フローチャートそのものではない．図 10 をフローチャートとみなして逐次的に実施することも，依存関係を満たした上で並列的に実施することも可能である.

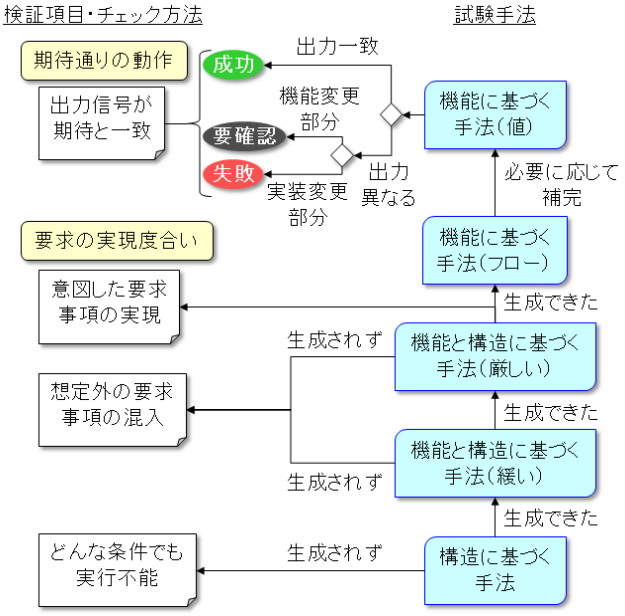

図 10　試験手法の適用方法

Figure 10　How to apply testing methods to verification cases

図 10 に示した試験手法の定義を表 6 に示す．この中で構造に基づく手法および機能と構造に基づく手法では，内部的な構造の実行可否を確実に求める必要があるため，形式手法の一種である有界モデル検査（Bounded Model Checking，BMC）を用いる.

表 6　試験手法の定義

Table 6　Definition of testing methods

試験手法	説明	具体例
機能に基づく手法 (値)	外部的な機能仕様の中で, 出力の期待値との一致を基準にする方法.	Back-to-Back テスト
機能に基づく手法 (フロー)	外部的な機能仕様の中で, コードの処理フローに影響を与える入力値を基準にする方法.	代表値組合せ
構造に基づく手法	内部的な構造の実行可否を基準にする方法. 例えば，分岐網羅や MC/DC など.	分岐網羅
機能と構造に基づく手法 (厳しい)	機能に基づく手法と構造に基づく手法の基準を同時に満たす方法の中で, 機能に関する基準が相対的に厳しいもの.	境界値分析 & 分岐網羅
機能と構造に基づく手法 (緩い)	機能に基づく手法と構造に基づく手法の基準を同時に満たす方法の中で, 機能に関する基準が相対的に緩いもの.	同値分割 & 分岐網羅

3.3.2 に従って生成する機能テストケースの全体集合は，各入出力条件と構造識別箇所の総当たり組合せである．この全体集合から選択した機能テストケースが，2.1 に示した試験工程に対する要求事項を満たした時点で図 6 の自動化フローを終了することで，組合せ爆発による生成失敗を回避する．自動化フローの終了条件を以下に，これを満たした状態の一例を図 11 に示す.

- 各入出力条件に対して, 少なくとも一つの機能テストケースが存在（○印）.
- 構造識別箇所に挿入した到達信号に対して, 少なくとも一つのテストケースが存在（○印）.
- 想定外の要求事項が混入した箇所を特定（△印）.
- どんな条件でも実行不能の箇所を特定（×印）.

	ただ一つの信号が有効境界値	ただ一つの信号が無効境界値	…
到達信号 A	○	−	−
到達信号 B	−	−	○
到達信号 C	−	○	−
…	△	△	△
…	×		

図 11　自動化フローの終了状態の一例

Figure 11　Termination condition example of automated flow

3.3.4 検証の実施方法

Simulink のシミュレーション機能を利用して，以下の手順で検証を実施する．これは図 5 の中の②に対応する．(1)(5)では MATLAB，(2)～(4)では Simulink を用いる．

(1) 前処理
(2) 検証シナリオを制御対象モデルに取込み
(3) 検証項目を制御対象モデルに設定
(4) アプリケーションと制御対象モデルを周期的に起動
(5) 後処理

4. 制御ソフトウェアへの適用例と施策評価

4.1 適用例と機能網羅・構造網羅の結果

適用の題材とする制御 S/W は，派生機種や仕向け地の違いによるバリエーションの増加が見込まれている．そこで，製品品質確保と省リソースの両立を狙ってモデルベース開発の導入を検討しており，主要な 6 機能の中の 3 機能について，Simulink を用いてモデル図面を作成している．モデル図面は Simulink の文法で自由に記述するのではなく，記述内容・表記法を規定することで，設計品質の均質化を図っている．モデル図面の記述内容・表記法の規定の中で，本方式の適用にも関係する入出力信号の明確化に関する規定の一例を図 12 に示す．

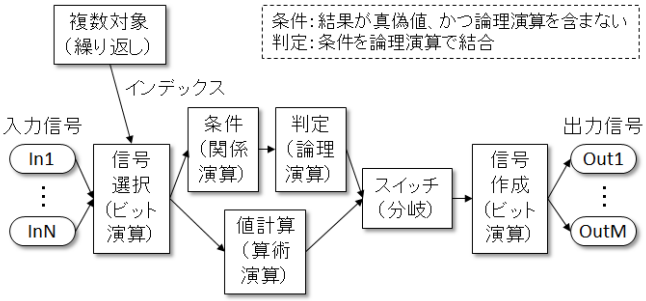

図 12 モデル図面の記述内容・表記法の規定例

Figure 12 Guideline example for modeling control software

今回モデル図面を作成した 3 機能の諸元を表 7 に示す．今回の適用例では，各種試験手法の中から最も基本的なものを取り上げ，機能に基づく手法は同値分割と境界値分析の組合せ（以降，同値・境界値），構造に基づく手法は分岐網羅とする．

表 7 題材とする制御 S/W の諸元

Table 7 Specification of target control software

項目	機能 A	機能 B	機能 C
入力信号数	15	12	9
出力信号数	1	2	2
入出力条件数 (同値・境界値)	46	31	24
構造識別箇所数 (分岐箇所)	34	60	30

ここで，同値分割と境界値分析を組み合わせる根拠を補足する．一般に，境界値分析の利用方法は信号が一つの場合で説明されることが多く，信号が複数の場合は明確ではない．そこで，例えば信号が一つの場合の説明を二つの場合に単純に当てはめると以下のようになり，各信号が有効境界値と無効化境界値を少なくとも一度は取るという基準は満たしている．

(ア) 信号 a が有効境界値，かつ，信号 b が有効境界値
(イ) 信号 a が無効境界値，かつ，信号 b が無効境界値

しかし，上記の方法には以下の不都合がある．

- S/W が実際に動作するときは，「すべての信号が境界値」を取るという状況は少なく，テストケースとして不足している．「ただ一つの信号が境界値，残りの信号は範囲」とする方がよい．
- 「すべての信号が無効境界値」を取るのは，無効条件はただ一つにするという試験手法の基本に反する．「ただ一つの信号が無効境界値，残りは有効範囲」とする方がよい．

以上より，信号が複数の場合は同値分割と境界値分析を組み合わせ，境界値分析のみよりも条件を弱めるのが妥当と考える．例えば，信号が二つの場合は以下のようになる．

(ア) 信号 a が有効境界値，かつ，信号 b が有効範囲．
(イ) 信号 a が有効範囲，かつ，信号 b が有効境界値．
(ウ) 信号 a が無効境界値，かつ，信号 b が有効範囲．
(エ) 信号 a が有効範囲，かつ，信号 b が無効境界値．

機能 A～機能 C の仕様として指定した入出力条件を表 8～表 10 示す．なお，製品の仕様情報を含む具体的な値は伏せている．

表 8 入出力条件－機能 A

Table 8 Input/output conditions on function A

信号	条件種別	条件式
入力 a	ビット	使用ビットの位置
入力 b	〃	〃
入力 c	値の範囲	最小値，最大値
入力 d	ビット	使用ビットの位置
入力 e	〃	〃
入力 f	〃	〃
入力 g	〃	〃
入力 h	論理値	なし
入力 i	論理値	なし
入力 j	値の列挙	列挙値のリスト
入力 k	〃	〃
入力 l	値の範囲	最小値，最大値
入力 m	〃	〃
入力 n	〃	〃
入力 o	〃	〃
出力 a	値の列挙	列挙値のリスト

表 9　入出力条件－機能 B

Table 9　Input/output conditions on function B

信号	条件種別	条件式
入力 a	ビット	使用ビットの位置
入力 b	〃	〃
入力 c	〃	〃
入力 d	〃	〃
入力 e	〃	〃
入力 f	値の列挙	列挙値のリスト
入力 g	値の範囲	最小値，最大値
入力 h	〃	〃
入力 i	〃	〃
入力 j	〃	〃
入力 k	〃	〃
入力 l	〃	〃
出力 a	ビット	使用ビットの位置
出力 b	〃	〃

表 10　入出力条件－機能 C

Table 10　Input/output conditions on function C

信号	条件種別	条件式
入力 a	ビット	使用ビットの位置
入力 b	〃	〃
入力 c	値の列挙	列挙値のリスト
入力 d	値の範囲	最小値，最大値
入力 e	〃	〃
入力 f	〃	〃
入力 g	〃	〃
入力 h	ビット	使用ビットの位置
入力 i	〃	〃
出力 a	ビット	使用ビットの位置
出力 b	〃	〃

機能 A～機能 C に本方式を適用した結果として得られた機能テストケースの諸元を表 11 に示す．ここで，題材とする制御 S/W は，ある処理周期での演算結果を状態値として保持して次以降の処理周期の演算で使用するという特徴を持つので，得られた機能テストケースも複数ステップで構成される入力信号列となった．

表 11　自動生成した機能テストケースの諸元

Table 11　Results of functional test case generation

項目		機能 A	機能 B	機能 C
テストケース数の合計		78	47	27
	1 ステップ入力	39	28	17
	2 ステップ入力	1	0	0
	3 ステップ入力	38	19	10

得られた機能テストケースを用いて，表 5 に示した検証項目の「要求の実現度合い」をチェックした結果を表 12～表 14 に示す．いずれの機能においても機能網羅と構造網羅がともに 100%になることを確認した．なお，「期待通りの動作」をチェックした結果は省略する．

表 12　検証項目のチェック結果－機能 A

Table 12　Result of checking verification case on function A

項目	入出力条件 (同値・境界値)	構造識別箇所 (分岐箇所)
意図した要求事項の実現	46/46	34/34
想定外の要求事項の混入	0/46	0/34
どんな条件でも実行不能	0/46	0/34
網羅度	100%	100%

表 13　検証項目のチェック結果－機能 B

Table 13　Result of checking verification case on function B

項目	入出力条件 (同値・境界値)	構造識別箇所 (分岐箇所)
意図した要求事項の実現	31/31	59/60
想定外の要求事項の混入	0/31	0/60
どんな条件でも実行不能	0/31	1/60
網羅度	100%	100%

表 14　検証項目のチェック結果－機能 C

Table 14　Result of checking verification case on function C

項目	入出力条件 (同値・境界値)	構造識別箇所 (分岐箇所)
意図した要求事項の実現	24/24	30/30
想定外の要求事項の混入	0/24	0/30
どんな条件でも実行不能	0/24	0/30
網羅度	100%	100%

機能 A～機能 C の機能テストケースの生成処理にかかった実時間，プロセッサ時間と実時間の比率，PC 諸元を以下に示す．生成処理はマルチコアを活用して並列化している．

- 実時間：13.5 分
- プロセッサ時間と実時間の比率：4.0
- CPU：Intel Core i7-3770 3.4GHz 8Core，メモリ：16GB

4.2 施策実現による想定効果の評価

4.1 の結果から，施策実現により開発対象 S/W を以下の基準でもれなく細分化することができた．

(a) 意図した要求事項の実現
機能仕様として指定した入出力条件を満たす機能テストケースを自動生成し，これだけを用いて構造識別箇所を実行できることを確認した．

(b) 想定外の要求事項の混入
今回の題材では該当するものはなかったが，意図した要求事項の実現とどんな条件でも実行不能のいずれでもない構造識別箇所を判別できることを別途確認している．

(c) どんな条件でも実行不能
従来手法を用いて該当する構造識別箇所を特定できることを確認した．これはコードであればデッドコードに相当するものである．

また，今回の題材では入出力条件として同値・境界値，構造識別箇所として分岐箇所を指定したが，検証手法としては表 6 に定義した試験手法に分類されるものであれば容易に拡張できるようにしている．そのため，プロジェクトで必要とされる試験手法が異なっても，各試験手法の基準を満たす機能テストケースを生成することができると考える．

以上より，2.3 で想定した「十分な試験項目一式を決定できる」という効果が見込めると判断する．

5. 関連研究

5.1 機能網羅と構造網羅の個別保証

機能に基づく網羅と構造に基づく網羅を別々に保証する研究を以下に示す．

- 橋本祐介，中島 震，ソフトウェアモデル検査とテストケース生成の統合ツールチェイン[7]
- 丁 明珍，特許 4924188 号，日本国特許庁．

これらの研究では，まず機能に基づく網羅を保証した後，未実行の構造識別箇所について機能仕様とは関係なく任意の値で実行するテストケースを求める．

この方法では，未実行箇所について機能網羅と構造網羅を同時保証するためのテストケースにはならないため，適切なテストケースを人手作業で補わなければならない．

5.2 有界モデル検査による構造網羅

有界モデル検査ツールの一つである CBMC（Bounded Model Checking for ANSI-C）は，ANSI-C 準拠の C 言語関数を静的解析し，オーバーフローや 0 除算などの不具合候補を検出する[8][9]. ツール内部で形式検証の一種である SAT solver を用いている．また，ESBMC（Efficient SMT-Based Context-Bounded Model Checker）は SMT solver を用いて整数型や浮動小数点型の変数や演算を直接扱うことで，CBMC より処理効率を向上している[10].

これらのツールは，ユーザが記述する assert 文のチェック機能を，構造に基づく網羅の保証に応用する研究がある．しかし，機能に基づく網羅の保証への応用は見当たらない．

6. おわりに

本稿では，大規模化・複雑化・細分化が急速に進む制御 S/W 開発の生産性向上の取り組みとして，開発コスト上昇の主要因となっている試験作業を効率化する機能テストケース生成方式を提案した．

具体的には，試験工程に対する要求事項を機能に基づく網羅と構造に基づく網羅の同時保証と規定し，これを実現するための各種試験手法の適用方法および機能テストケースの自動生成方法を考案した．また，実開発に導入する際に試験作業を妨げないことを考慮して，自動生成にかかる時間を短縮する方法も盛り込んだ．

題材とする制御 S/W への適用例と施策評価から，施策実現により得られると想定した「十分な試験項目一式を決定できる」という効果が見込めると判断する．

今回の適用例では各種試験手法の中から最も基本的な同値分割・境界値分析と分岐網羅を取り上げた．検証手法としては各種試験手法に容易に拡張できるようにしており，今後は各種試験手法に適用範囲を拡大して評価を進め，本方式の有用性をさらに向上させていく予定である．

参考文献

[1] Klaus Pohl, Günter Böckle, Frank van der Linden, *ソフトウェアプロダクトラインエンジニアリング ソフトウェア製品系列開発の基礎と概念から技法まで*, エスアイビー・アクセス, 2009.
[2] Kevin Forsberg, Hal Mooz, Howard Cotterman, *Visualizing Project Management 3rd Edition*, Wiley, 2005.
[3] 西村秀和，“モデルベースでシステムズエンジニアリングを推進するための SysML,” *SysML 利活用協議会設立記念フォーラム* (2014).
[4] *DO-178B Software Considerations in Airborne Systems and Equipment Certification*, RTCA, Inc.,1992.
[5] *Road vehicles -- Functional safety -- Part 1-10*, ISO, 2011-2012.
[6] Kelly J. Hayhurst, Dan S. Veerhusen, John J. Chilenski, Leanna K. Rierson, *A Practical Tutorial on Modified Condition/Decision Coverage*, NASA/TM-2001-210876, 2001.
[7] 橋本祐介，中島 震，“ソフトウェアモデル検査とテストケース生成の統合ツールチェイン,” *ソフトウェアエンジニアリングシンポジウム 2011* (2011).
[8] Damiano Angeletti, Enrico Giunchiglia, Massimo Narizzano, Alessandra Puddu, Salvatore Sabina, “Automatic Test Generation for Coverage Analysis of ERTMS software,” *ICST 2009* (2009).
[9] Edmund Clarke, Daniel Kroening, Flavio Lerda, “A Tool for Checking ANSI-C Programs,” *TACAS 2004, vol.2988 of LNCS* (2004).
[10] Lucas Corderio, Bernd Fischer, Joao Marques-Silva, “SMT-Based Bounded Model Checking for Embedded ANSI-C Software,” (2011).