一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
SS2016-3,KBSE2016-9(2016-07)

組込みソフトウェアを対象とした状態遷移表抽出手法

山本　椋太† 　吉田　則裕† 　竹田　彰彦†† 　舘　　伸幸† 　高田　広章†

† 名古屋大学大学院情報科学研究科　〒 464–8601 愛知県名古屋市千種区不老町
†† 組込みシステム技術協会　〒 103–0011 東京都中央区日本橋大伝馬町 6–7
E-mail: †yamamoto.ryota@k.mbox.nagoya-u.ac.jp, yoshida@ertl.jp, n-tachi@nagoya-u.jp, hiro@ertl.jp
††takeda@opt-net.jp

あらまし　レガシー化した組込みソフトウェアは理解することが困難になっており，保守や再利用に大きなコストがかかる．リアルタイム制御に行う組込みソフトウェアは状態遷移モデルで表現できる．そのため，状態遷移の理解および不整合検出のために状態遷移表を用いることができる．本研究では，レガシーコードから状態遷移表を抽出することでソフトウェア理解を支援する手法を提案する．
キーワード　リバースエンジニアリング，コード解析，状態遷移表

Extracting State Transition Tables from an Embedded Software System

Ryota YAMAMOTO†, Norihiro YOSHIDA†, Akihiko TAKEDA††, Nobuyuki TACHI†, and Hiroaki TAKADA†

† Graduate School of Information Science, Nagoya University
Furo-cho, Chikusa-ku, Nagoya, Aichi, 464–8601 Japan
†† Japan Embedded Systems Technology Association
6-7, Odenma-cho, Chuo-ku, Tokyo, 103–0011 Japan
E-mail: †yamamoto.ryota@k.mbox.nagoya-u.ac.jp, yoshida@ertl.jp, n-tachi@nagoya-u.jp, hiro@ertl.jp
††takeda@opt-net.jp

Abstract　It is hard to understand legacy code for an embedded software system. It leads much cost for maintaining and reusing the system. Real-time control for an embedded software system can be represented as a state transition model. Developers are able to use the state transition model for understanding the state transitions of the system and detecting inconsistency from the transitions. In this study, we propose an approach to extracting state transition tables from legacy code for the understanding of an embedded software system.
Key words　Reverse-Engineering, Code Analysis, State Transition Table

1. はじめに

組込みシステムの高度化は著しく，開発の大規模化，複雑化が進んでいる．それに対して，高品質・低コスト・短納期によるソフトウェア開発の要求がある [1]．そのため，組込みシステム開発の現場ではソフトウェアの再利用性や保守性の向上の要求がある．しかし，ソースコードのレガシー化によって，再利用や保守におけるコストが増大している現状がある [2]．

現在の開発現場においては，設計書が存在しないレガシーコードも存在しうる．そのため，ソースコードをブラックボックスとしてしか扱うことができず，このような状態でレガシーコードを扱うことは，ソフトウェアの品質低下につながる [2]．

組込みシステムはリアルタイム制御システムであることが多いため，イベント・状態・動作・遷移が多く存在しており [3]，イベント・状態・動作・遷移を網羅的に理解することは困難である [2]．

加えて，現在の開発現場においては，設計書が存在しないレガシーコードも存在しうるため，ソースコードをブラックボックスとしてしか扱うことができない場合がある．このような状態でレガシーコードを扱うことによってソフトウェアの品質低下につながり，ソフトウェアの再利用手法を適用することが困難となる [2]．

状態遷移表とは，事象 (イベント)・状態・処理 (動作)・遷移からなる普遍的なモデルである．状態遷移表は，システムが

ある状態のときに事象が発生すると，どのような処理が行われ，どのような状態に遷移するのかを表す．状態遷移表でシステムを表現することのメリットは，システム全体を網羅的に俯瞰することができる点である．その結果，ある状態において事象が発生したときの処理についての欠落を発見することができる．

本稿では，組込みソフトウェアを対象とした，レガシーコードから状態遷移表を抽出する手法を提案する．また，提案手法は手作業によって適用可能な手法であるが，ツール化の検討も行う．実装したツールの評価のため，ツールを小規模の組込みソフトウェアに対して適用する．

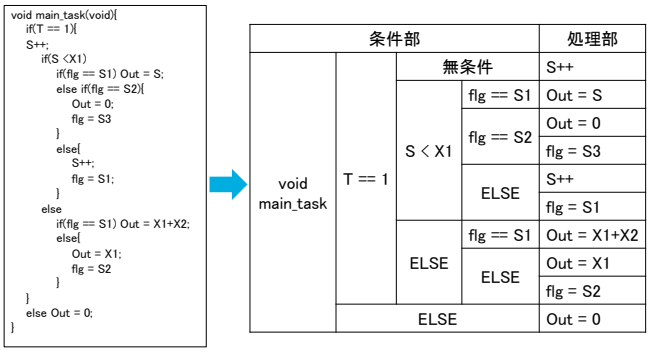

図 1　条件処理表の例
Fig. 1　Example of Condition-Process Table

2.　提 案 手 法

本章では，手作業によって状態遷移表を出力するための方法を説明する．手作業における手法を基としたツール化の方法に関しては，次章で説明する．以下，手作業における提案手法の流れを示す．

（1）　ソースコードから条件処理表を作成する．
（2）　条件処理表から状態変数を選択する．
（3）　選択した状態変数に関する状態遷移表を作成する．

また，ここで入力されるソースコードは，コメント文が削除され，マクロ展開を行った C 言語のソースコードであることとする．

上記の手順について，次節以降にて説明する．

2.1　条件処理表

まず，条件処理表について説明する．条件処理表とは，ある条件において実行される処理を関数ごとにまとめた表である．本稿においては，状態遷移表を抽出するための中間状態として作成される．条件処理表の作成例を図 1 に示す．条件処理表は，図 1 のように条件部と処理部に分割される．

条件部は，ソースコード内の関数および条件分岐文 (if, elseif, else および switch) から作成される．条件部について，ソースコードから条件処理表に変換するための具体的な変換手順について説明する．まず，if および elseif については，その条件式が条件部に記述される．else の場合は，表に ELSE と記述することとする．switch については，switch(var) のように指定された変数 var と，case num:のようにラベルで指定された値 num を用いて，表に var == num のように記述する．ラベルが default の場合については，ELSE として表記する．

処理部は，ある条件部に対応する処理が記述される．

2.2　状態変数の選択

条件処理表を作成した後，条件処理表から状態変数を見つけ，全ての状態変数の中から，状態遷移表で表現したい状態変数を選択する．手動による手法においては，状態変数を複数選択してもよい．

ここで，状態変数の定義は，以下のとおりである．

- 状態変数は分岐条件に使用されている．
- 状態変数が取りうる値は有限個である．
- 状態変数は必ず内部で更新される．

具体的には，変数 var について，if(var == 1) のような条件分岐文を考える．その条件分岐文のスコープの中で，var の値が更新されているなら，var が状態変数になりうる．

たとえば，図 1 の条件処理表における状態変数候補は，変数 flg である．変数 flg は，条件式に含まれており，かつそのスコープ内で値が更新されているため，flg を状態変数として選択することができる．

ただし，状態変数は必ずしも 1 つの変数になるとは限らない．if(a == 1 && b == 0) のような条件分岐文では，その条件分岐文のスコープの中で，a か b，またはその両方の値が更新されているなら，a && b が状態変数になりうる．

2.3　状態遷移表の作成

選択された状態変数について，状態遷移表を作成する．状態遷移表の作成は，条件処理表を基として行う．

まず，条件処理表の処理部の変更について説明する．処理部において，状態変数が取りうる値の分だけ条件処理表の処理部の列を複製する (図 2)．

変数 flg は，S1，S2 および ELSE の 3 つの値を取りうるため，同じ処理部を 3 列作成し，各列を，flg が S1，S2 および ELSE のそれぞれの値の場合に取りうる処理に対応させる．すると，flg が S1 のときには，flg が S2 であるときの処理が実行されないといったように，実行されない処理がいくつか存在しうる．

次に，条件処理表の条件部の変更について説明する．条件部において，flg に関わる条件が処理部の上部に移動したため，flg == S1 などの条件式は不要となる．このように，不要な箇所が，図 2 中において，黒く塗り潰されている．これらの不必要な箇所を削除し，表を整形すると，図 3 のように，整形された状態遷移表を出力することができる．

そして，今回の例では適用対象にならないが，選択した状態変数を含む条件式よりも深いネストの条件式は，条件部から除外し，代わりに処理部に統合することとする．

以上の手順によって，C 言語のソースコードから，状態遷移表を出力する．

3.　自動抽出ツールの実装

本章では，提案手法を実装するために実施した作業を説明する．以下に作業の流れを示す（括弧内は実装言語である）．

条件部				flg		
				S1	S2	ELSE
void main_task	T == 1	無条件		S++	S++	S++
		S < X1	flg == S1	Out = S	Out = S	Out = S
			flg == S2	Out = 0	Out = 0	Out = 0
				flg = S3	flg = S3	flg = S3
			ELSE	S++	S++	S++
				flg = S1	flg = S1	flg = S1
		ELSE	flg == S1	Out = X1+X2	Out = X1+X2	Out = X1+X2
			ELSE	Out = X1	Out = X1	Out = X1
				flg = S2	flg = S2	flg = S2
	ELSE			Out = 0	Out = 0	Out = 0

図 2　状態遷移表への変換

Fig. 2　Transforming into State Transition Table

条件部			flg		
			S1	S2	ELSE
void main_task	T == 1	無条件	S++	S++	S++
		S < X1	Out = S	Out = S	S++
			–	Out = 0	flg = S1
			–	flg = S3	–
		ELSE	Out = X1+X2	Out = X1	Out = X1
			–	flg = S2	flg = S2
	ELSE		Out = 0	Out = 0	Out = 0

図 3　状態遷移表の例

Fig. 3　Example of State Transition Table

（1） C 言語のソースコードを一定の形式に整形し，構文解析する．(GCC + Python 3.4)

（2） 構文解析の結果から，条件処理表を作成する．(Excel VBA)

（3） 条件処理表から状態変数を選択するための UI を実装する．(Excel VBA)

（4） 選択された状態変数に関する状態遷移表を作成する．(Python3.4 + Excel VBA)

2. において説明した手順と比べ，C 言語のソースコードを一定の形式に整形する作業が追加されている．この手順を実施することで，構文解析における負担を低減することができる．また，実装環境について，Python3.4 および Excel VBA を使用している．

上記の手順について，次節以降にて説明する．

3.1　対象ソースコードに対する構文解析

本節では，解析対象のソースコードの整形およびその構文解析の方法を述べる．以下にその大まかな流れを示す．

（1） 不要な#include の削除を行う．標準ライブラリを始めとしたヘッダファイルの include を削除し，(2) においてマクロ展開される際の出力が簡単になるようにする．削除対象のヘッダファイルについて，ユーザがあらかじめ指定することもできる．

（2） GCC によるマクロ展開およびコメントの削除を行う．構文解析する際には，コメント文が不必要であるため削除している．また，構文解析の際に，定数かどうかの判断が複雑になるため，マクロ展開をしている．

（3） プログラムの形式の整形およびインデントの削除を行う．具体的には，中括弧の位置の統一や，中括弧が存在しない条件分岐文への中括弧の追加を行っている．インデントの削除の目的は，(4) において条件分岐文に識別子を付与する際に，余分な空白文字を削除しておく必要があることである．また，インデントの形式がソースコードによって異なるため，統一形式でのインデントにするべく，インデントを一旦削除することとしている．

（4） 条件分岐文に対する識別子の付与を行う．ソースコード中の if，else if，else，switch，case および default に識別子を付与している．この識別子を用いて，条件式の構造解析を行う．

（5） 条件式を “無条件” とした if 文の追加を行う．この目的は，条件処理表の条件部において，“無条件” と表示することである．

（6） インデントの追加を行う．インデントは，ネストの深さを判断するために使用される．しかし，インデントは削除されているため，この段階でインデントの追加を行う．インデントの追加は，中括弧を基準に行われる．

（7） グローバル変数・関数プロトタイプの宣言，#pragma およびインラインアセンブラの削除を行う．グローバル変数・関数プロトタイプの宣言については，今後の処理に不必要であるため削除する．インラインアセンブラは，今回の解析においては対象外としているため，削除する．

（8） 状態変数候補を抽出する．状態変数の定義に従い，状態変数を抽出する．まず，変数をすべて抽出し，抽出した変数について，(i) 条件分岐文における条件式に抽出した変数が存在するか，(ii) 該当の条件分岐文のスコープ内で，抽出した変数が 1 度でも更新されているか，(iii) 抽出した変数は他の変数と比較されておらず，その取りうる値が有限であるか，これら 3 つの条件にすべて当てはまるものを状態変数候補として抽出する．

（9） 不要な関数の削除を行う．(8) において抽出した，状態変数候補を含まない関数を削除する．その理由は，状態変数候補を含まない関数は，条件処理表において状態変数の選択を検討するための材料にならないためである．ただし，後の実装において，関数のインライン展開を行う可能性を考え，例外として，状態変数候補を含む関数から呼び出されている関数は，状態変数候補を含まないとしても，削除しない．

（10） 条件分岐文の構文解析を行う．ある条件分岐文が持っている処理文と条件分岐文を解析する．その結果として，条件分岐文のスコープ内にある処理文と条件分岐文の構造を得る．得られた構造データから，Excel の Sheet におけるセルの結合情報を得る．このとき，列アドレスについてはネストの深さを，行アドレスについては条件分岐文の構造から，条件分岐文をすべて処理文に展開したときの処理文の行数を，それぞれ使用して求めている．

（11） TSV 形式の条件処理表を出力する．構文解析の結果から，“セルの始点アドレス”，“セルの終点アドレス”，“セルの値” となるように TSV ファイルを出力する．

上記のように，対象ソースコードに対して，構文解析を行っ

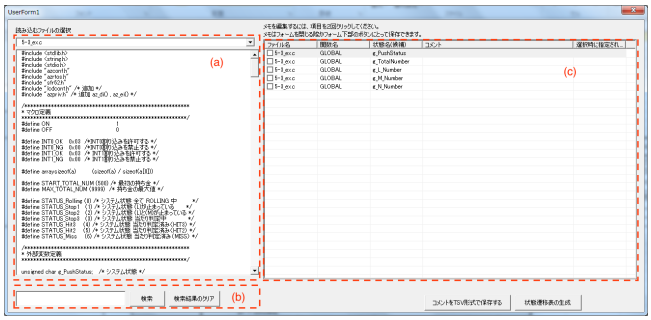

図 4　状態変数候補選択 UI
Fig. 4　User Interface for Selecting Condition Variables

ている．

3.2　条件処理表の出力

本節では，構文解析の結果，出力された TSV 形式の条件処理表から，Excel の Sheet 上への条件処理表の出力方法を述べる．

上述の通り，“セルの始点アドレス”，“セルの終点アドレス”，“セルの値” というフォーマットの TSV ファイルが存在している．Excel VBA において，その TSV ファイルの各行をタブ文字ごとに分割し，分割された “セルの始点アドレス” から “セルの終点アドレス” の範囲のセルを結合する．そして結合されたセルに，“セルの値” を書き込む．

以上によって，条件処理表が完成する．

3.3　状態変数の選択

条件処理表を作成した後は，状態変数候補の中から状態変数を選択し，状態遷移表を出力する．このとき，状態変数の選択を自動的にツールが行うのではなく，ユーザが行うものとする．その理由は，あるソースコードには複数の状態変数候補が存在し，状態変数としていずれの状態変数候補を選択すれば有効な条件処理表を作成できるかの判定が，ツールにとって困難であるためである．

そこで，Excel VBA によって，状態変数候補を選択するための補助を行うことを目的とした，フォームアプリケーションを作成する．フォームアプリケーションの概観を図 4 に示す．

以下，フォームアプリケーションの機能について説明する．

(a)　ソースコードプレビュー機能

元のソースコードを表示する機能である．状態変数を探す際に，元のソースコードを確認必要がある場合などに，活用することができる．

(b)　状態変数検索機能

状態変数名をテキストフォームに入力し，検索することができる機能である．もし，検索結果がヒットした場合，該当行がハイライトされる．

(c)　状態変数選択機能

状態変数候補の中から，状態変数として選択する変数を選択することができる機能である．コメントを残す機能も搭載しており，変数の行をダブルクリックすることで，コメントを記入することができる．

最後に，“状態遷移表生成” のボタンをクリックすると，状態遷移表の出力処理を行う．以上のように，状態変数選択のための UI を実装した．

3.4　状態遷移表の出力

前節にて選択された状態変数についての状態遷移表を出力する．まず，状態遷移表における，選択された状態変数を含む条件分岐文は不要であるため，削除する．次に，処理文について，選択した状態変数が取りうる値に対して，それぞれ，実行される処理と実行されない処理を検出する．実行されない処理は必要ないため，黒く塗りつぶす．実行される処理だけが見えるように，表を作成する．最後に，表を整理して状態遷移表の出力が完了する．

4.　ケーススタディ

本章では，手動およびツールによる提案手法の適用結果を述べる．このとき，適用対象とするソースコードはいずれも同じものを使用している．この対象とするソースコードは，状態変数を持っており，if および else といくつかの処理からなるシンプルで小規模なものである．加えて，コメント文およびマクロは存在しないものとする．ソースコードの規模に関して，LOC が 141 行，条件分岐文が 24 文，そして処理文が 68 文である．各適用結果においては，状態遷移表を示すものとする．

手動による提案手法の適用結果について，状態遷移表を図 5 に示す．条件処理表の作成のために約 20 分，表中に存在する 2 つの状態変数候補の発見のために約 5 分，そして状態変数候補の中からある変数についての状態遷移表の作成するために約 28 分必要であったため，あわせて約 53 分を必要とした．

このように，手動による適用実験においては，141 行程度の規模のソースコードに対して 53 分程度必要である．本来，手動による提案手法の適用においては，ソースコード中のコメント文の削除や，マクロの展開などの作業が必要になるため，さらに多くの時間を必要とする．

次に，手法の限界について考察する．提案手法の目的は，組込みソフトウェアにおける状態変数に対する状態遷移表の出力である．そのため，リアルタイム OS 上で動作するアプリケーションを想定した場合，リアルタイム OS におけるセマフォなどのオブジェクトの遷移に対する状態遷移表の出力を行うことができず，アプリケーションの状態を理解するための情報が不足する可能性がある．

また，ツールによる提案手法の適用結果について，状態遷移表を図 6 にそれぞれ示す．これらを作成には，2 分程度が必要であった．表の生成を，それぞれ 30 秒程度の時間で行うことができ，状態変数候補も，ツールによってすでに提示されているため，1 分程度で状態変数を選択し，状態遷移表を出力することができた．

ツールによる適用実験では，非常に短時間で手動の場合と同様の結果を得ることができた．また，手動で適用手法を適用した経験から，ツールの機能の中でも状態変数候補をツールが自動的に提案する機能は，ソースコードが複雑化するにつれ，非常に有用であるように感じた．このことから，手法のツール化は，作業コストの大きな削減につながると言え，手法

				vNLIMF	
				0	ELSE
fcal_mode3	vovr_mode == 3			fcal_mode3judgement();	fcal_mode3judgement();
	ELSE			vNLIMF = 0;	vNLIMF = 0;
				vTLIMF = 0;	vTLIMF = 0;
				vAPSIN3F = 0;	vAPSIN3F = 0;
				vAPSIT3F = 0;	vAPSIT3F = 0;
				vm3_status = 0;	vm3_status = 0;
fcal_mode3judgement	vTLIMF == 0			if(vAPST30 < vAPSN30){	if(vAPST30 < vAPSN30){
				if(0.0f < vAPSN3){	if(0.0f < vAPSN3){
				vNLIMF = 0;	vNLIMF = 0;
				vTLIMF = 0;	vTLIMF = 0;
				vAPSIN3F = 0;	vAPSIN3F = 0;
				vAPSIT3F = 0;	vAPSIT3F = 0;
				vm3_status = 1;	vm3_status = 5;
				}	}
				else{	else{
				vNLIMF = 1;	vNLIMF = 1;
				vTLIMF = 0;	vTLIMF = 0;
				vAPSIN3F = 1;	vAPSIN3F = 1;
				vAPSIT3F = 0;	vAPSIT3F = 0;
				vm3_status = 2;	vm3_status = 6;
				}	}
				}	}
				else{	else{
				if(0.0f < vAPST3){	if(0.0f < (vAPST3+vAPSN3)){
				vNLIMF = 0;	vNLIMF = 0;
				vTLIMF = 0;	vTLIMF = 0;
				vAPSIN3F = 0;	vAPSIN3F = 0;
				vAPSIT3F = 0;	vAPSIT3F = 0;
				vm3_status = 3;	vm3_status = 7;
				}	}
				else{	else{
				vNLIMF = 0;	vNLIMF = 0;
				vTLIMF = 1;	vTLIMF = 1;
				vAPSIN3F = 0;	vAPSIN3F = 0;
				vAPSIT3F = 1;	vAPSIT3F = 2;
				vm3_status = 4;	vm3_status = 8;
				}	vmode3_nsumz = vmode3_nsum;
				}	}
				—	}
	ELSE	vAPSN30 < vAPST30	0.0f < vAPST3	vNLIMF = 0;	vNLIMF = 0;
				vTLIMF = 0;	vTLIMF = 0;
				vAPSIN3F = 0;	vAPSIN3F = 0;
				vAPSIT3F = 0;	vAPSIT3F = 0;
				vm3_status = 9;	vm3_status = 9;
			ELSE	vNLIMF = 0;	vNLIMF = 0;
				vTLIMF = 1;	vTLIMF = 1;
				vAPSIN3F = 0;	vAPSIN3F = 0;
				vAPSIT3F = 1;	vAPSIT3F = 1;
				vm3_status = 10;	vm3_status = 10;
		ELSE	0.0f < (vAPST3 + vAPSN3)	vNLIMF = 0;	vNLIMF = 0;
				vTLIMF = 0;	vTLIMF = 0;
				vAPSIN3F = 0;	vAPSIN3F = 0;
				vAPSIT3F = 0;	vAPSIT3F = 0;
				vm3_status = 11;	vm3_status = 11;
			ELSE	vNLIMF = 1;	vNLIMF = 1;
				vTLIMF = 0;	vTLIMF = 0;
				vAPSIN3F = 2;	vAPSIN3F = 2;
				vmode3_tsumz = vmode3_tsum;	vmode3_tsumz = vmode3_tsum;
				vAPSIT3F = 0;	vAPSIT3F = 0;
				vm3_status = 12;	vm3_status = 12;

図 5　手作業による状態遷移表の作成

Fig. 5　Manually-generated State Transition Table

をツール化することによる効果を確認することができた．

ツールの限界について考察する．現在のツールが対応しているソースコードは，今回の適用実験にて用いたソースコードのように，簡単な構造のものだけである．そのため，さらに複雑な構造のソースコードに対応する必要がある．

手動による手法とツールによる手法の，それぞれの適用における差について考察する．図 5 と図 6 において，表の内容は一致しており，異なる点は，表のサイズと，表の体裁である．

これらの差異について説明する．表のサイズが異なっている理由は，中括弧の位置が異なるためであり，表の体裁として，条件部が中央揃えか右揃えになっているか，および実行しない処理の表現方法が “-” であるかセルを黒く塗りつぶしているかの違いだけである．これらの差異は，表の内容に対してまったく影響を及ぼさない．

以上の通り，今回の適用実験においては，手動およびツールにおいて，同等の条件処理表および状態遷移表を得ることができた．そのため，小規模であり，かつ単純な条件分岐文のみから構成されるソースコードからは，状態遷移表を抽出することが可能であり，組込みソフトウェアから状態遷移表を抽出することができる可能性を見出すことができたと言える．

さらに検討事項として，

- ループ文の取り扱い
- 関数のインライン展開
- 状態変数を同時に 2 つ以上選択した場合の自動作成
- マクロ定数の扱い
- ELSE の範囲による状態変数の状態値のラベルの決定方法

上記が挙げられる．

				vNLIMF	
				0	ELSE
void fcal_mode3(void)				fcal_mode3judgement();	fcal_mode3judgement();
				vNLIMF = 0;	vNLIMF = 0;
				vTLIMF = 0;	vTLIMF = 0;
				vAPSIN3F = 0;	vAPSIN3F = 0;
				vAPSIT3F = 0;	vAPSIT3F = 0;
				vm3_status = 0;	vm3_status = 0;
void fcal_mode3judgement(void)				if(vAPST30 < vAPSN30)	if(vAPST30 < vAPSN30)
				{	{
				if(0.0f < vAPSN3)	if(0.0f < vAPSN3)
				{	{
				vNLIMF = 0;	vNLIMF = 0;
				vTLIMF = 0;	vTLIMF = 0;
				vAPSIN3F = 0;	vAPSIN3F = 0;
				vAPSIT3F = 0;	vAPSIT3F = 0;
				vm3_status = 1;	vm3_status = 5;
				}	}
				else	else
				{	{
				vNLIMF = 1;	vNLIMF = 1;
				vTLIMF = 0;	vTLIMF = 0;
				vAPSIN3F = 1;	vAPSIN3F = 1;
				vAPSIT3F = 0;	vAPSIT3F = 0;
				vm3_status = 2;	vm3_status = 6;
				}	}
				}	}
			vTLIMF==0	else	else
				{	{
				if(0.0f < vAPST3)	if(0.0f <(vAPST3+vAPSN3))
				{	{
				vNLIMF = 0;	vNLIMF = 0;
				vTLIMF = 0;	vTLIMF = 0;
				vAPSIN3F = 0;	vAPSIN3F = 0;
				vAPSIT3F = 0;	vAPSIT3F = 0;
				vm3_status = 3;	vm3_status = 7;
				}	}
				else	else
				{	{
				vNLIMF = 0;	vNLIMF = 0;
				vTLIMF = 1;	vTLIMF = 1;
				vAPSIN3F = 0;	vAPSIN3F = 0;
				vAPSIT3F = 1;	vAPSIT3F = 2;
				vm3_status = 4;	vm3_status = 8;
				}	vmode3_nsumz = vmode3_nsum;
				}	}
				[redacted]	}
	ELSE	vAPSN30 < vAPST30	0.0f < vAPST3	vNLIMF = 0;	vNLIMF = 0;
				vTLIMF = 0;	vTLIMF = 0;
				vAPSIN3F = 0;	vAPSIN3F = 0;
				vAPSIT3F = 0;	vAPSIT3F = 0;
				vm3_status = 9;	vm3_status = 9;
			ELSE	vNLIMF = 0;	vNLIMF = 0;
				vTLIMF = 1;	vTLIMF = 1;
				vAPSIN3F = 0;	vAPSIN3F = 0;
				vAPSIT3F = 1;	vAPSIT3F = 1;
				vm3_status = 10;	vm3_status = 10;
		ELSE	0.0f <(vAPST3 + vAPSN3)	vNLIMF = 0;	vNLIMF = 0;
				vTLIMF = 0;	vTLIMF = 0;
				vAPSIN3F = 0;	vAPSIN3F = 0;
				vAPSIT3F = 0;	vAPSIT3F = 0;
				vm3_status = 11;	vm3_status = 11;
			ELSE	vNLIMF = 1;	vNLIMF = 1;
				vTLIMF = 0;	vTLIMF = 0;
				vAPSIN3F = 2;	vAPSIN3F = 2;
				vmode3_tsumz = vmode3_tsum;	vmode3_tsumz = vmode3_tsum;
				vAPSIT3F = 0;	vAPSIT3F = 0;
				vm3_status = 12;	vm3_status = 12;

図 6　ツールが出力した状態遷移表

Fig. 6　State Transition Table Extracted by the Proposed Tool

5. おわりに

本稿では，組込みソフトウェアを対象とした状態遷移表の抽出手法を提案した．

また，提案手法のツール化を行い，実装したツールを小規模の組込みソフトウェアに対して適用した結果，状態遷移表を抽出することができた．このことから，提案手法のツール化の可能性があると言える．また，手動による提案手法の適用に比べて，数パーセント程度の時間で状態遷移表を抽出することができ，自動化の効果を確認することができた．

今後の課題として，ツールによって状態遷移表を抽出することができるソースコードの構造を増やし，状態変数を複数選択した場合の状態遷移表を出力することができるようにするなどの機能追加を含めた，さらなるツールのブラッシュアップを行う必要がある．

文　献

[1] 高田広章，“組込みシステム開発技術の現状と展望，” 情報処理学会論文誌，vol.42，no.4，pp.930–938，2001.

[2] 竹田彰彦，“状態遷移表によるレガシーコードの蘇生術，” http://techon.nikkeibp.co.jp/article/COLUMN/20150519/418967/.

[3] 独立行政法人 情報処理推進機構ソフトウェア・エンジニアリング・センター，組込みソフトウェア開発における品質向上の勧め［設計モデリング編］，アイティメディア株式会社，2006.