Vol.2012-SLDM-155 No.26
Vol.2012-EMB-24 No.26
2012/3/3

SLDS 機構を用いた SystemC-Verilog HDL トランスレータの開発

佐藤賢文† 三井浩康†

近年，企業間ではシステムレベル設計言語を用いた HW/SW 協調設計の普及が進んでいる。しかし，教育機関での普及は使用するツール（例：動作合成ツール）が高価であるといった背景により，途上である。本研究では，学生がシステムレベル設計言語を用いた HW/SW 協調設計を学ぶために，動作合成ツールの代替として，SystemC-Verilog HDL トランスレータを開発する。トランスレータ開発のためにフロントエンド，ミドルエンドは新規に開発し，バックエンドは Design Methodology Lab 提供の論理合成 CAD ツール「Simple Logic Design System(SLDS)」の機構を利用した。

Development of SystemC–Verilog HDL Translator Using SLDS Mechanism

MASAFUMI SATO† HIROYASU MITSUI†

In recent years, the HW/SW co-design method using system-level design language has become popular in many enterprises. However, the education in universities has not been developed enough due to the expensive cost of using tools (example: Behavior Synthesis Tool). In this paper, the SystemC-Verilog HDL Translator is proposed and developed as substitute for a expensive behavior synthesis tool, in order to help students learn HW/SW co-design method using system-level design language in universities. For developing translator, front-end and middle-end were developed newly in this research and Simple Logic Design System (SLDS), which is a logic synthesis CAD tool supplied by Design Methodology Lab was used for the back-end of this translator. The C language was used for developing the translator. In the front-end, the mechanism that analyzes SystemC and saves analysis result into the syntactic tree was developed. In the middle-end, the mechanism that generates HPLS language based on the syntactic trees, which are generated in the front-end, was developed, in order to use the mechanism of SLDS that can convert HPLS language to Verilog HDL.

1. はじめに

近年，特定の機能を実現するためのコンピュータシステムである組込みシステムは大規模化，複雑化が進み，新しい設計手法が求められている．従来方式では先に HW 開発を行い，その後に SW 開発を行うが，SW 開発の大規模化と共に開発期間の圧迫が問題になっている．

こうした問題を解決する新しい設計手法として，近年 HW/SW 協調設計が注目されている．HW/SW 協調設計ではシステム設計の早期に HW 部，SW 部の機能分割を行い，HW と SW を並行して設計，検証，開発を進める．従来方式と本研究で扱う HW/SW 協調設計方式の比較を図 1 に示す．

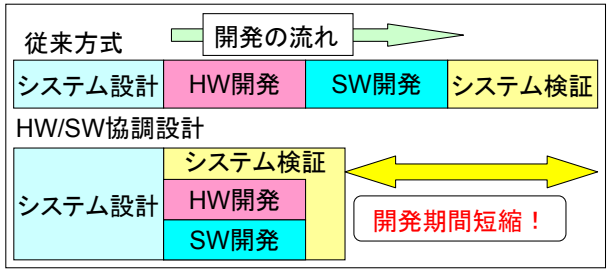


図 1 従来方式と HW/SW 協調設計方式の比較
Figure 1 Comparison of traditional method and HW/SW co-design method

HW/SW 協調設計は企業の開発現場では普及が進みつつあるが，教育機関での教育，学習は盛んではない．その理由の一つとして，開発環境，及びツールが高価なことが挙げられる．アルゴリズムから HW 記述を生成する動作合成ツールを例に挙げると，単体でも数十万円から数百万円になる．このような背景から，大学の研究室等で学生が HW/SW 協調設計を手軽に学ぶことが困難な現状がある．

2. 関連研究

HW/SW 協調設計に関しては様々な研究が行われている．動作合成手法は既に確立さ

† 東京電機大学大学院理工学研究科
Graduate School of Science and Engineering, Tokyo Denki University

Vol.2012-SLDM-155 No.26
Vol.2012-EMB-24 No.26
2012/3/3

れた技術であり，動作合成の活用方法に狙いを定めた研究が多い．「仕様記述言語による RTL 記述の生産性の試行評価」[1]の研究では，動作合成ツールを導入し，SpecC 言語を用いて高位設計から詳細設計までを行い，HW 記述言語を用いた設計との比較，評価を行っている。SpecC を用いた場合，シミュレーションを高速で行えると共に，HW 記述言語を用いた際とほぼ同量の機能記述に納められることを示した．

動作合成ツール自体の開発を行っている研究は稀である．例として，「上位ハードウェア設計言語 Melasy+による VHDL コード生成と動作検証」[2]の研究では，C++言語を拡張した上位 HW 設計言語 Melasy＋を入力として，複数の HW 記述言語の出力を可能とするコンパイラ開発を行っている．

3. 研究目的

本研究では，学生が実機を用いた SystemC による HW/SW 協調設計を学ぶために，動作合成ツールの代替品となるツールを開発することを目的とする．開発するツールは SystemC 記述を元に HW 記述言語の Verilog HDL を生成するトランスレータである．

4. 研究で使用する技術

4.1 SystemC[3]

SystemC はシステムレベルの設計，検証用のシステムレベル設計言語の一つである．SystemC は普及推進・標準化団体である Open SysmteC Initiative(OSCI)が無償配布している．SystemC は C++のクラスライブラリの一つと定義されており，文法は C++に依存する．また，システムの設計段階に応じた抽象度モデルでシステムを記述できる．

システム仕様を高抽象度で機能記述し，システム設計の上流でシミュレーション可能なモデル（UTF・TF モデル）を提供できる．このモデルを元に HW，SW の機能割り振り（トレードオフ）を早期に行えるので，図 2 に示すように HW と SW の平行開発が可能である．トレードオフ後は BCA モデル，RTL モデルの順に詳細化を行う．

本研究では RTL モデルを設計対象とする. RTL モデルは設計するシステムの HW モジュールに対して，機能記述，モジュール間通信等の全動作をシステムクロック同期で記述するモデルである．SystemC を用いた HW/SW 協調設計によるシステム設計のフローを図 2 に示す.

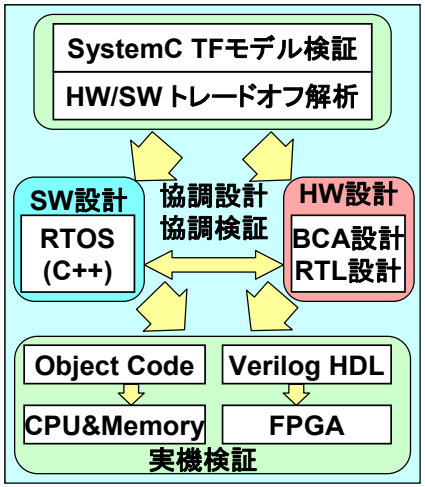


図 2　SystemC を用いた HW/SW 協調設計の流れ
Figure 2　Flow of HW/SW co-design method with SystemC

4.2 トランスレータ[4]

トランスレータは言語処理系の一つで，入出力が同抽象度であることが特徴である．機能面から，フロントエンド，ミドルエンド，バックエンドに分割される．

(1)　フロントエンドは入力ソースコードの言語仕様に応じて字句解析, 構文解析を行う．
(2)　ミドルエンドは構文解析の結果を元に中間コードの生成と最適化を行う．
(3)　バックエンドは出力コードを生成する．

本研究で開発するツールは SystemC の RTL モデルを元に Verilog HDL（RTL 記述）を生成するので，トランスレータに分類される．

4.3 Simple Logic Design System(SLDS)

SLDS は，Design Methodology Lab[5]提供のディジタル回路設計システムである(6)．ブロックダイアグラムと呼ばれるグラフィカルな回路入力，または独自の HW 記述言語である Logic Description Format(以下，LDF)を入力として，一旦中間コード Hewlett Packard Logic Synthesize System（以下，HPLS)に変換後，Verilog HDL を出力する機構を持つ．本研究ではこの LDF を元に Verilog HDL を出力する機構を，提案するトランスレータのバックエンドとして利用する．

Vol.2012-SLDM-155 No.26
Vol.2012-EMB-24 No.26
2012/3/3

4.4 HPLS

HPLS は，SLDS の中間コードに採用されている，Hewlett Packard 社が開発した HW 記述言語及び，それを含めたネットリスト生成システムである．HPLS では基礎的な論理ゲートが提供されており，ユーザはそれらの論理ゲートを用いて HW を構築する．それぞれのゲート記述は一律した文法規則に則っており「.回路名(出力リスト)(入力リスト)」で構成される．例を挙げると，2 入力（A，B）と 1 出力（C）を持つ論理積演算を HPLS で記述すると，「 .AND(C)(A B) 」と記述される．

本研究では開発するトランスレータのバックエンドとして 4. 3 で述べた SLDS を利用する．SLDS のバックエンドの SLDS の入力は 4．4 で述べた HPLS なので，ミドルエンドはフロントエンドから得た構文解析結果を元に HPLS を生成する．

5. 研究内容

5.1 SLDS 機構を用いた SystemC-Verilog HDL トランスレータの開発

図 3 に開発するトランスレータの内部構成図を示す．

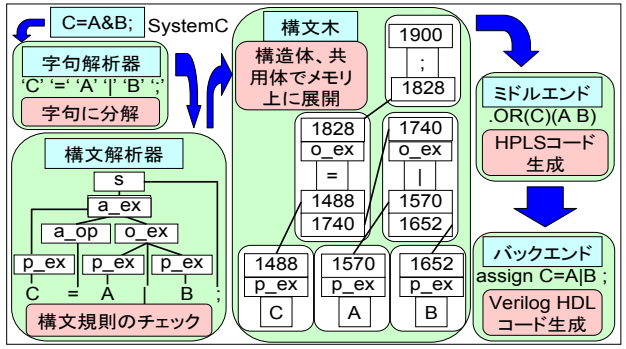


図 3　トランスレータの入力から出力までの流れ
Figure 3　Input-output flow of the translator

SystemC の RTL モデルを入力し，Verilog HDL による RTL 記述を出力するトランスレータを開発する．

フロントエンドは SystemC ソースコードを元に，字句解析，構文解析を行う．構文解析器の要求を元に字句解析器が動作し，言語仕様上最小限の字句単位に分割し，構文解析器に渡す．構文解析器はそれを元にソースコードの構文が正しいかチェックし，構文構成をメモリ上にリスト構造で展開してミドルエンドに渡す．

ミドルエンドはフロントエンドから渡された構文木に保存された情報を元に，バックエンドの入力となる中間コードの HPLS を生成する．

バックエンド部分は，SLDS の機構を利用し，実際にはフロントエンドと，ミドルエンドの設計，開発を行う．

実装する機能は以下に示す論理回路の SystemC 記述を Verilog HDL 記述に変換可能とする．

- 組み合わせ回路
- 順序回路
- 単一モジュール構成システム
- 複数プロセスシステム
- 階層構成システム

SystemC の初心者向け教本[3]に載っている内容を検討して，上記の回路，システム記述を扱えれば，学生が用いるツールとして十分な機能を持つと考える．

5.2 フロントエンドの開発

字句解析器の開発には flex，構文解析器の開発には bison を用いた．flex，bison は Linux 系システムに標準で搭載されているコンパイラ・コンパイラである．共に C 言語と独自の記述で表現した解析器をコンパイルして，記述した通りに動作する C 言語プログラムを出力する．また，flex，bison 共に，相互に連携して動作することを前提に開発されており，flex-bison 間インターフェース用の変数や関数，マクロが用意されている．[6]

字句解析器は入力ソースコードの言語仕様を元に字句で切り分け，構文解析に必要なトークン番号，トークン種別を割り振り構文解析器に渡す役割を持っている．

構文解析器は字句解析器の解析結果を受けて，ソースコードが構文規則に則って記述されているかチェックを行う．構文解析の結果得られた情報は，プログラム全体の構成を表す構文木の形で保存される．

構文解析器の実装において SystemC 独自の記述方式と C 言語記述の両方を解析できる必要があったため，C 言語記述の構文規則は ANSI 規格を参考に作成を行い，SystemC 記述の構文規則は独自に生成した．

Vol.2012-SLDM-155 No.26
Vol.2012-EMB-24 No.26
2012/3/3

開発したフロントエンド内の構成を，主要モジュールを用いて図 4 に示す．

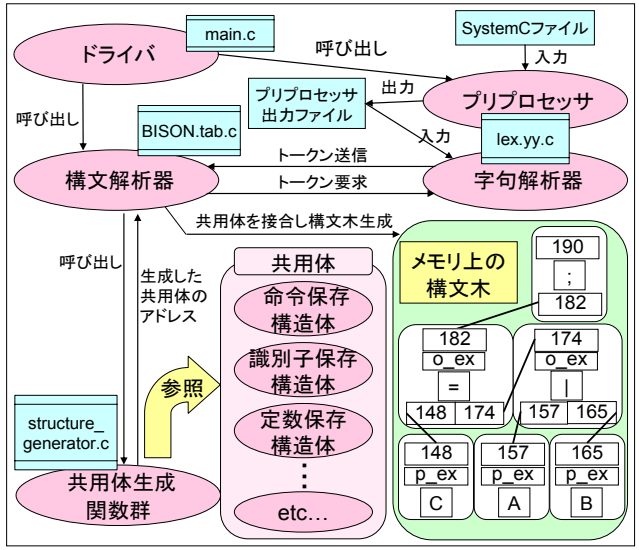


図 4 フロントエンド内主要モジュール構成図
Figure 4　Configuration diagram of the primary modules in front-end

main.c はドライバとして動作し，フロントエンド内各モジュールの動作を制御する．

lex.yy.c は flex によってコンパイルされたプログラムであり，字句解析器として動作する．また，ソースコードとトランスレータのインターフェースとなるので，機能を拡張してプリプロセッサ機能の一部として動作する．

Bison.tab.c は bison によってコンパイルされたプログラムであり，構文解析器として動作する．構文解析の結果は，structure_generator.c の共用体生成関数群にアクセスすることで生成された共用体のアドレスを元に，各共用体をポインタで接続してメモリ上に構文木として残す．

共用体は複数の構造体型をまとめている．共用体でどの構造体型が扱われているかは，共用体の持つタグによって確認することができる．それぞれの構造体は SystemC の「命令に関する情報を保存する」，「識別子に関する情報を保存する」等の用途がそれぞれ決まっている．structure_generator.c の共用体生成関数群は構文解析器から呼び出され，構文解析の進捗に合わせて適宜共用体を生成し，生成した共用体のアドレスを構文解析器に送り返す．

生成された構文木は葉（構文木の末端）に字句解析結果の字句に関する情報を持ち，葉と根の間の節点に構文規則に関する情報を持つ．葉や節点の情報は構造体，共用体を用いて保存し，葉と節点，節点同士を結ぶ枝は，ポインタで接続する．フロントエンドで生成された構文木はミドルエンドの入力となり，構文木上に保存された各データが HPLS 生成のための情報源となる．

5.3　ミドルエンドの開発

ミドルエンドは構文木を元に HPLS を生成する．そのための機構を大きくわけると，ex 関数，スケジューリング関数群，HPLS 生成関数群がある．ex 関数は再帰関数であり，構文木を構成する各ノードへのアクセスに用いられる．再帰関数 ex は，構文木を構成する各ノードが保持するポインタを元に，一度実行されると全てのノードを走査する．各ノードに到着時，そのノードの保持情報に応じてスケジューリング関数群を呼び出す．スケジューリング関数群は出力の制御を行い，SystemC 文法上，出力タイミングをずらす必要がある命令等を解決する．図 5 に，ex 関数の呼び出しを元にスケジューリング関数郡が動作する例を示す．

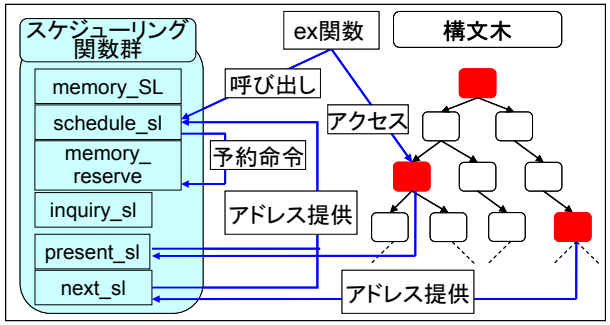


図 5　スケジューリング関数群の動作例
Figure 5　Semantics of scheduling function group

図 3 では ex 関数が呼び出したスケジューリング関数群の関数 schedule_sl によって，現

在 ex 関数がアクセスしている構文木のノード以降に保存された情報を，next_sl で指定された構文木のノードの前段に移す予約を行っている．この処理後，ex 関数が next_sl に到達した時点で inquiry_sl によってこの予約処理が行われ，構文木の構成が変化することで後の出力が変化する．

スケジューリング関数等による構文木の操作が終了した後，ex 関数によって HPLS 生成関数群が呼び出される．ex 関数が構文木の走査を行い，アクセスした構文木ノードに保存された情報に従って，HPLS 生成関数群を呼び出す．HPLS 生成関数郡は ex 関数から渡された構文木ノードの情報を元に，HPLS ソースの生成源となる構造体を生成する．この生成された構造体のメンバ変数に HPLS 生成関数群は値を代入する．構造体は一つで HPLS 一行分の情報量を持つ．生成された構造体は構造体ポインタ配列で一括管理されており，HPLS 生成群はこの配列にアクセスして構造体に保存された情報を元に HPLS を生成する．図 6 に，HPLS 生成関数群が ex 関数に呼び出されて動作する例を示す．

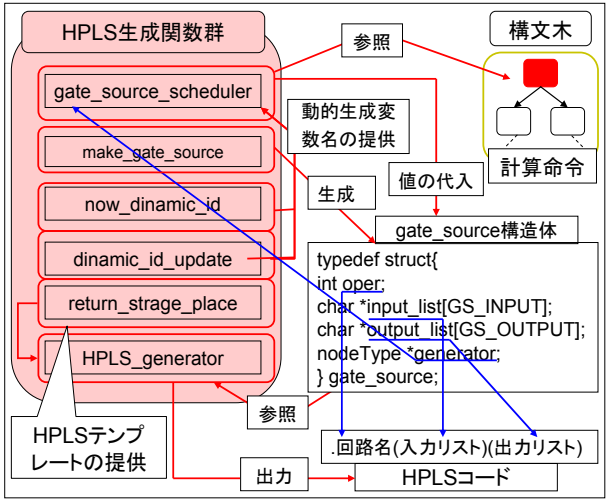


図 6　HPLS 生成関数群の動作例
Figure 6　Semantics of HPLS generating function group

図 6 では，ex 関数が計算命令を保持する構文木ノードにアクセスすることで生成された HPLS 出力用の情報を保持する gate_source 構造体に対し，構造体の各メンバに値を入力し，その構造体を元に HPLS を一行出力するまでのフローを示している．

gate_source 構造体は make_gate_source 関数によって生成され，gate_source 型ポインタ配列 gate_source_list によって一括管理される．make_gate_source 関数は ex 関数の構文木走査時に，各計算命令を保存したノードに行き着いた際に呼び出され，必要な個数の gate_source 構造体を生成する．また，構文木ノードのアドレスを gate_source 構造体のメンバ変数 generator に，構文木ノードに保存された計算に用いたオペレータを gate_source 構造体のメンバ変数 oper に保存する．ex 関数の走査終了後，main 関数によって gate_source_scheduler 関数が呼び出される．

gate_source_scheduler は ex 関数を模した関数で，構文木の各ノードに保存された情報を元に gate_source 構造体の各メンバの値を代入していく．SystemC から HPLS に変換する際，両言語の仕様上，識別子が足りなくなるケースがある．その際には，now_dinamic_id 関数，dinamic_id_update 関数から動的に足りない分の識別子を生成して取得し，構造体のメンバに格納する．この操作を gate_source_scheduler 関数が構文木ノード全てを走査するまで繰り返すことで，HPLS を出力するための情報を得る．

全ての構文木ノード走査が終了後，HPLS_generator 関数により，gate_source 構造体リストを元に HPLS が生成される．gate_source 構造体のメンバはそれぞれ，oper が HPLS における回路名，input_list が回路の入力元，output_list が回路の出力先を示している．HPLS_generator 関数は return_strage_place より HPLS 出力テンプレートを得る．HPLS 出力テンプレートは char 型二次元配列に格納されており，return_strage_place 関数は HPLS_generator 関数より渡された gate_source 構造体のメンバ変数 oper の値に応じてこの配列にアクセスして HPLS 出力テンプレートを HPLS_generator 関数に渡す．出力テンプレートの例を示すと，AND 回路のテンプレートは「.AND( )( )」である．HPLS_generator 関数は gate_source 構造体の各パラメータをテンプレートに組込み HPLS を生成する．

生成された HPLS は，SLDS のバックエンドに入力される．SLDS には HPLS を Verilog HDL に変換する機構が備わっているため，その機構を利用して Verilog HDL を出力する．

5.4　評価

実際に SystemC コードをトランスレータに入力し，入力 SystemC コードと開発したミドルエンドの出力結果である HPLS，ミドルエンドから得られた HPLS を SLDS に入力することで得られる Verilog HDL のそれぞれの比較を行う．ここでは例として，基本論理回路である AND，OR，NOT に関する計算を行う SystemC モジュールとその機能記述を

フロントエンドに入力し，出力された HPLS，Verilog HDL と比較した図を図 7 に示す．

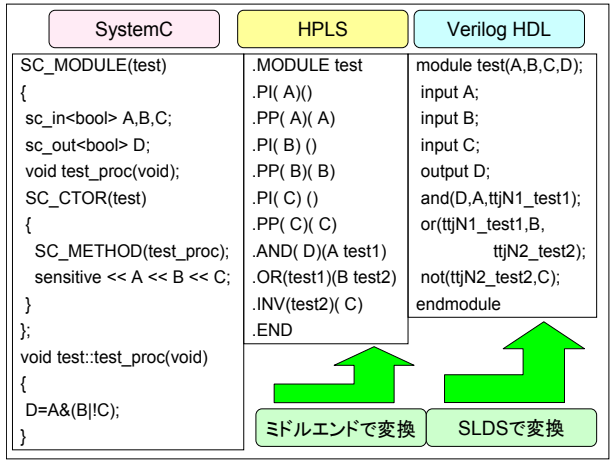


図 7　SystemC，HPLS，Verilog HDL の比較
Figure 7　Comparison of SystemC, HPLS and Verilog HDL

SystemC から HPLS に変換する際に，test1，test2 といった変数が増えている．これは HPLS の文法上, HPLS で提供されている論理ゲートでは SystemC の計算命令 D=A&(B|!C) を一行で記述できず，複数行に分ける必要が生じたためである．この処理を 5．3 で述べた HPLS 生成関数群で行うことで，SystemC と HPLS の等価性を保っている． HPLS から Verilog HDL への変換はバックエンドとして利用している SLDS の機能であり, Verilog HDL のプリミティブ・ゲート（言語仕様として提供されている，基本的なゲート群）を利用することで，HPLS との等価性を保っている．

6.　まとめと今後の課題

SLDS 機構を用いた SystemC-Verilog HDL トランスレータの開発に関して説明した．今後は機能の拡充を行い，5．1 で述べた目標とするトランスレータの機能の全範囲をカバーできることを目標とする．

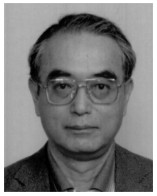
謝辞　本研究を進めるにあたって，Design Methodology Lab の田中基夫氏には SLDS をバックエンドに使用するにあたり，的確なアドバイスと情報を提供していただき、ありがとうございました。また，研究室各位には様々な場面で研究を支えていただきました。皆様に感謝の意を表します。

参考文献

[1]八田佳憲，泉知論，吉川寿広，荒木大: 仕様記述言語による RTL 記述の生産性の試行評価，(2011)
[2]白鳥航亮， 和崎克己: 上位ハードウェア設計言語 Melasy+による VHDL コード生成と動作検証, (2010)
[3]並木秀明， 後簡哲也， 片岡忠士: SystemC による System デザイン入門”, 技術評論社， Vol2005, pp12-21， pp78-338 (2005)
[4]宮本 衛市 ： はじめてのコンパイラ 原理と実践， Vol2007， pp25-94， (2007)
[5]Design Methodology Lab
http://www.methodologylab.com/Design_Methodology_Lab/HOME.html
[6]THOMAS NIEMANN　: A Guide To Lex & Yacc，ePaperPress

著者紹介

佐藤賢文（学生会員）2010 年東京電機大学理工学部情報システム工学科卒業，現在，東京電機大学大学院理工学研究科情報学専攻在学中．学生の組込みシステム設計・開発に関しての学習を支援するソフトウェア開発，実験方式の提案等に従事．電気学会会員．



三井浩康（正会員）1967 年東京大学工学部電気工学科卒業．同年三菱電機(株)入社．空港管制システム，情報システム・機器などの研究開発に従事．2000 年より東京電機大学に勤務．現在特任教授．組込みシステムの研究に従事．博士（情報学），電子情報通信学会，電気学会，IEEE-CS, ES 各会員．