Vol. 35 No. 8　　情　報　処　理　　Aug. 1994

解　説

論理設計の形式的検証

2. 形式的検証手法の実設計への適用例†

藤田昌宏††　陳　奔†††　山崎正実††††

1. はじめに

設計対象が大規模・複雑化するにつれ，設計対象の動作の正しさを論理シミュレーションのみで確認することは，シミュレーションすべきパターン数の多さから，きわめて困難になりつつあると言われている．また，複雑な動作をするシステムに対し，必要十分なシミュレーション用テストパターンを人手で作成することは不可能に近い．実際，設計者が作ったテストデータは設計者の思うとおりに動作することを確認するためのものが多く，設計者が予期していなかったような動作は調べられていないことが多い．ところが，実際のバグはほとんどそのような予想されていなかった場合に起因している．

さらに，実際に設計・製造したチップにバグが発見された場合，その原因をシミュレーションで解析するのは容易ではない．実際，われわれは，ある通信用のチップ（以降，このチップの名前をA Buggy Chip, ABC とする）の設計・製造において，次のような状況に陥った．ABC は入力2ポートからデータを受けとり，いったんバッファに蓄え，制御信号の指示に従って，順序を必要に応じて変えながら，出力の2ポートのうち，指示されたポートに出力していく．当然，入力されたデータはいつかは出力されなければならないし，また，入力されたデータが失われたり，同じデータが複数回出力されるようなことがあってはならない．ところが，設計を終了し（もちろん，シミュレーションで設計の正しさを確信し），製造して実際のフィールドテストを行ってみると動作がおかしいことに気がついた．それも，最初の数秒は正しく動作するが，その後，入力されたデータが複数回出力されたり，消滅したりすることが確認された．ABC は 156 MHz で動作しているため，数秒間に 1G サイクル程度動作しており，とてもシミュレーションで再現できない．シミュレーションで原因を探るには，かなりの工夫が必要と思われ，効果的とは言いがたい．

幸い，CAD 部門では，最新の研究成果を迅速に取り込みたいということから，形式的検証に関する一般的技術についての調査に一人アサインされていた（彼の名前を仮にXとする）．Xはすでに，1カ月くらい形式的検証技術一般について勉強してきており，ちょうど，何かよい例題はないかを探している時期だった．そこで，形式的検証手法を応用して，なんとかバグの原因を探れないかという課題がXに課されることになった．Xは，本特集の解説「論理関数処理に基づく形式的検証手法」で説明されている記号モデル検査手法を利用して，異常動作の原因を追求していき，結果的に入力データが複製されたり，消滅する原因を発見することに成功した．本稿では，彼がいかにして原因を突き止められたかを説明していく．これによって，「形式的検証を実際に応用した場合，どれだけ有効で，また，どのような問題に直面するか」を理解する手がかりとなれば幸いである．

なお，本稿で取り上げる問題は，形式的検証ツールを利用して，設計のバグの原因をいかに見つけるかであり，形式的検証のツールの一般的な利用法である，設計の正しさを証明することとは目的が異なる．ここでは，設計者がバグの理由を理解すればそれで目的が達成されたと考える．このため，設計をかなり抽象化して検証している．

† A Trial Use of Formal Verification Techniques for Bug Identification of a Real Chip by Masahiro FUJITA (Fujitsu Laboratories of America), Ben Chen (Fujitu Digital Technology) and Masami YAMAZAKI (Fujitsu).

†† 米国富士通研究所
††† 富士通デジタルテクノロジ
†††† 富士通

通常の形式的検証では，この設計の抽象化自体が正しいかどうかが問題となるが，ここでは，設計者がバグの原因を理解できる限り，設計の抽象化の正しさは問わない*.

まず，**2.** では，ABC の概要について説明する．次に **3.** では，使用した形式的検証ツール (SMV と呼ばれる[3]) の概要の説明と，ABC のモデル化の仕方について説明する．また，X がいかに SMV を理解していったかについても簡単に触れる．**4.** では，バグを探っていった実際の過程について述べ，バグを発見するまでの流れについて説明する．次の **5.** では，本経験から得られた形式的検証手法を実設計に利用する上でのポイントを整理する．最後に **6.** で結論を述べる．

2. 検証対象

設計記述が仕様を満たすことを証明することが検証である．設計記述は，ゲートレベルの回路として与えられた．また，仕様に関しては，X が設計者に聞く形で以下のようにまとめられた．X にとっては，まずこの仕様の理解が第一の仕事となる．

2.1 仕　様

ABC の動作のイメージは**図-1** を用いて，以下のように説明できる．

入力ポートである iHW0/iHW1 から取り出したデータが Frame Buffer にいったん保存され，必要に応じて出力ポート oHW0/oHW1 に出力される．Frame Buffer へデータを書き込む際，アドレスの制御は WC (Write Control) と WAF (Write Address FIFO) で行われる．一方，Frame Buffer からデータを出力するには，二つの RAF (Read Address FIFO) の一方からアドレスをもらい，RC (Read Control) を経由してデータの読みだしを制御する．これと同時に，使われなくなった Frame Buffer のアドレスを回収し，WAF に送り，次のデータを保存するために用意しておく．

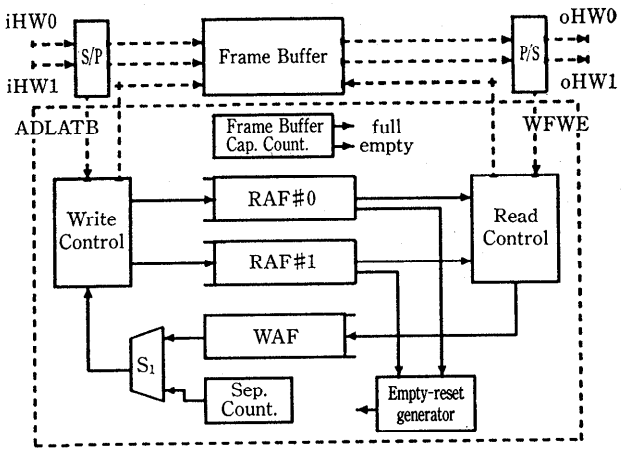


図-1 ABC 内部構造

電源を入れた直後は，WAF 部と RAF 部にアドレスデータがないため，最初のアドレスデータは SC (Sequential Counter) で生成される．Frame Buffer のすべてのアドレスの生成が終わったら，SC は切り離され，WAF から回収されるアドレスを循環的に使う．一方，二つの RAF (RAF#1/RAF#0) の容量が 0 の場合（要するに，Frame Buffer にデータのない状態），これは "Empty-reset" と呼ばれる状態となり，ふたたび SC からアドレスを順次生成してもらい，電源を入れた直後の動作を繰り返す．

WAF への書込みは信号 "WFWE" によって行われる．一方，WAF の読みだしは信号 "ADLATB" で制御される．書込みと読みだしの関係は非同期的である．なお，WAF の容量は FBCC (Frame Buffer Capacity Counter) で管理される．FBCC は，ADLATB でカウントアップ，WFWE でカウントダウンし，WAF の容量とは裏返しの条件で動作しているため，FBCC の "Full" は WAF の "Empty" と，"Empty" は "Full" と解釈することができる．このため WAF は専用の容量カウンタをもたない．

3. 使用した形式的検証ツールとモデル化の概要

われわれの使用した形式的検証ツールは，CMU の Clarke 教授のグループが開発した SMV と呼ばれるプログラム[3] で，仕様を CTL と呼ばれる時相論理で記述し，二分決定グラフ[1),4)] を利用した記号モデル検査法[2)] によって検証するもの

* このチップは結局，再設計し製造し直さなければならず，その場合には通常，機能拡張も行われる．したがって，その再設計の参考になればよく，その意味で，設計者がバグの理由を理解することが最も重要である．一方，通常の検証の場合には，設計の抽象化が間違っていると，何を検証しているのか分からなくなり，この抽象化過程を検証することもきわめて重要である．

である．CTL や記号モデル検査法に関しては，本特集の解説「論理関数処理に基づく形式的検証手法」[5] を参照していただくこととし，ここでは特に説明せず，SMV の受けつける設計記述に関してのみ簡単に説明する．

SMV は，状態遷移表現の設計記述を受けつける．

それは，たとえば，論理設計の教科書によく出てくる「自動販売機」の動作記述のようなものを状態遷移で表現できる．また，複雑な組合せ回路がある場合には，それを状態遷移記述とは別に与えることもできるし，さらに階層記述も可能である．このため，階層的に表現された回路図も基本回路を状態遷移表現に変換することにより，ほぼそのまま SMV の言語に変換することができる．ただし，扱える回路規模の点で，このような直接変換は一般的には使えない．通常，SMV が検証できる設計規模とは，状態を表す変数（回路を直接変換する場合にはフリップフロップの数に相当する）が 50 程度までが限度であり，回路を直接変換して検証できるのは，フリップフロップ数が 50 程度以下の回路に限定されてしまう．このため，特殊な場合☆ を除いて，回路を直接変換しても検証できず☆☆，なんらかの「設計の抽象化」をまず施すことが必須である．

したがって，検証しようと思っている回路の規模が小さい場合には，回路をそのまま変換することにし，回路規模が大きくなると，検証したい仕様を考慮して設計を抽象化し，状態を表す変数の数を減少させる工夫を随時行っていった．

仕様は，設計記述に現れる変数を用いた CTL 式（時相論理式）の形で表現される．SMV は設計が仕様を満たすかを調べ，満たさない場合には，反例（状態遷移シーケンス）を出力してくれる．この反例を解析することによって，設計が仕様を満たさない原因を考えることができる．ただし，この反例の表示も初心者には分かりにくい面があり，今後の研究成果が期待される分野である．

☆ 制御回路や非同期回路のように，回路規模は小さいが，考慮しなければならない場合の数が多いような回路では，人手で発見できなかったようなバグを形式的検証手法を用いて数多く発見できている．

☆☆ 具体的には，検証途中で二分決定グラフの大きさが爆発的に増大し，処理できなくなる．

4. 検証過程

ここでは，Xが検証していった過程を順次説明していく．最終的に三つの異なるモデルを作成し，結果的にバグの原因を解析することができた．モデルの作成は設計者と頻繁に打合せを行いながら，以下の 2 点から対象を絞って検証していった．

1. 設計者があやしいと感じる部分（設計に自信のない部分）

2. フィールドでの異常動作に関連する考えられる部分

フィールドテストの結果から，入力されたデータが消滅したり，複製されたりする場合があることが分かっている．内部ではデータは Frame Buffer に蓄えられているため，Frame Buffer のどこを使用しているかを示すアドレスを管理している部分がおかしく，特定のアドレスが一部消滅したり，同じアドレスが複数個管理されていると考えられる．そこで，これに関連する部分で，設計者が最もあやしいと思った（つまり自信がない）FIFO から検証を進めていった．しかし，後述するように，結局は，設計者があやしいと考えた部分ではなく，設計者が考慮していなかった動作シーケンスが原因であることが分かり，形式的検証の重要性が改めて確認された形となった．

4.1 FIFO の検証

設計者があやしい（言い換えると自信がない）と感じたのが FIFO であるため，図-1 中の WAF をまず検証することになった．**図-2** に WAF の内部構造を示す．

WAF は高速化のため キャッシュレジスタを二つ（#0, #1）もつ FIFO である．データがキャッシュにある場合には，キャッシュの速度で動作するが，データがキャッシュレジスタにない場合には，RAM から直接出力されるため，速度が低下する．また，WAF にデータが 1 個以下しか

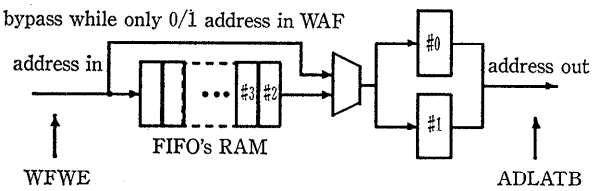


図-2 WAF の内部構造

ないときは，入力データは RAM をバイパスされ，直接キャッシュレジスタに格納される．信号 WFWE が1になると WAF にデータが格納され，信号 ADLATB が1になると WAF からデータが出力される．

この WAF が最もあやしい回路ブロックであることから，ゲート回路の記述をそのまま変換する形で SMV のための記述を作成した．しかし，WAF は8ビット幅で FIFO の深さが 168 あり，そのままでは状態変数の数が大きくなり過ぎるため，データ幅を1ビットにし，FIFO の深さも5に縮小したモデルを作成した☆．したがって，WAF は状態を6種類もつことになる (FIFO の深さ5とデータがまったくない状態を一つ加えたもの)．このようにすると，WAF に対する SMV のための記述は，計 22 個の状態変数をもつことになった☆☆．

このモデルに対し，調べた仕様は以下のとおりである．

1. WAF にデータがない状態から始めて，WAF にデータがいっぱいになる状態に辿りつくことがあり得るか☆☆☆？正常に FIFO が埋まっていけば，いっぱいになるはずである．

2. WAF にデータがない場合に，信号 ADLATB が1になることはないか☆☆☆☆？ データがないのに出力することになるので，ADLATB は決して1になってはいけない．

3. キャッシュレジスタと内部の RAM のデータのやりとりをする信号が，実際にデータ転送が行われるまで保持されているか☆☆☆☆☆？ これは当然保持されていなければならない．

これらの仕様のいずれに対しても，WAF は正しいことが SMV によって確認された．Xはここまでくるのに約1カ月かかっているが，それは SMV 自体やモデルの書き方，それに仕様の表現方法を理解するための時間が大半であり，検証時間自体はワークステーションで，数秒である．

SMV の設計記述は一般のハードウェア記述言語とある程度似ているが，少し記述の仕方が異なる部分もある．このため，ハードウェア設計者が幾分混乱することも少なくない．実際，Xも最初のうちは，自分の思っている動作を本当に書いているかを検証することに大半の時間を費やしていた．

WAF が正しいということになり，ではどうするかが問題となった．こうなると，図-1 中のアドレス管理に関係する部分全体が互いに関連して問題を起こしていると考えられるので，これら全体のモデルをなんとかして作成する必要がある．このときが，バグ解明の糸口がまだなく，ABC 全体をモデル化しないといけないということで，Xにとって最も辛い時期だったということになる．回路から動作を考え，それをある程度抽象化して SMV のための状態遷移記述を作成するのであるから，大変である．まあとにかく，簡単なモデルからということで，まず，アドレスを循環的に利用することに本質的に関連する部分を中心としてモデルを作成した．それをここでは状態モデルと呼ぶ．

4.2 状態モデル

アドレスの管理には，図-1 の WAF, RAF (二つ)，WC, RC が構成するループ，それにいくつかの制御回路 (SC, Empty-reset 制御回路など) が関係している．メモリなどを除いたランダムロジックで約8Kゲート程度である．回路規模からして，そのまま変換して SMV のためのモデルを作成できないことは明らかである．

さて，どのように抽象化（あるいは簡略化）したらよいか？まず，WAF はすでに検証して正しいと分かっている☆ので，この部分をできるだけ簡略化することにした．先ほどのモデルでは状態変数を 22 使用していたが，キャッシュレジスタをなくすなどして，それを9まで減らす簡略化を行った．RAF も FIFO であるため，同様に簡略化した．また，WC や RC は，実際にはさまざまな制御が入っているが，アドレスの循環的利用という立場から，単なるレジスタに置き換えた．また，empty-reset に関しては，実際の回路の意

☆ 状態遷移の関係を表現する二分決定グラフの大きさを見ながら，これらの大きさを決定した．具体的には，二分決定グラフの大きさが数万以下で収まるようにしたが，これは単に経験から決めた数である．

☆☆ 直観的には，データの幅や深さによって，検証結果が変わるとは考えにくいが，正式にはなんらかの方法で証明しておく必要がある．しかし，ここではバグの原因を設計者が理解することに重点を置いているので，直観的に正しいと思われる抽象化はそのまま利用している．

☆☆☆ CTL では EF 演算子を用いて表現できる．

☆☆☆☆ CTL では AGAF 演算子を利用して表現できる．

☆☆☆☆☆ CTL では AU 演算子で表現できる．

☆ より正確には試した仕様に関しては正しいと分かっている．試していないが満たさなければならない仕様もあるかもしれない．

味を考えて，二つの RAF が共に空になったときに1になるような制御にした．また，empty-reset が1になると，WAF も RAF もリセットされるようにした．データのビット幅はすべて1ビットにしている．このモデルは結局，23 の状態変数を使用している．

さて，仕様をどうするかであるが，結局のところ試行錯誤しかなかった．データパス系が1ビットであるため，データの値に関する仕様は記述できず，各種制御信号や FIFO の状態に関する CTL 式を試行錯誤的（つまり，理解している回路の動作から満たすべきだと考えられる仕様を順に）試すことを行っていった．そして，最後に次の仕様に辿りついた．empty-reset が1になると，アドレスは SC から供給されるようになるはずであり，WAF に新しくアドレスが到着するには WC, RAF, RC を経由することから（図-1 を参照されたい），少なくとも3サイクルはかかる．ということは，empty-reset が1になると，少なくとも3サイクルは WAF の状態は空から動かないはずである．これを CTL で記述し，検証してみたところ，反例が表示された．つまり，SC 以外のどこからかアドレスが供給されているのである．バグはここにある！この検証時間も1分程度であるが，ここまでくるのに約2カ月かかっている．

さて，バグを完全に解明するには，データパス系を最低2ビットにし，同じアドレスが重複するかどうかを調べる仕様を直接検証できるようにする必要がある*．しかしここまでくると，バグの解明はもう間近とXも感じるようになっており，ゴール目の前ということで，データパス系を2ビットにしたモデルを作成していった．これをデータパスモデルと呼ぶ．

4.3 データパスモデル

基本的には，データパスを2ビットにした状態モデルがデータパスモデルである．しかし，単純に2ビットにすると，状態変数の数が 70 になり，SMV で検証しようとすると，どうしても二分決定グラフのノード数が爆発してしまった．そこで，SMV の入力記述を順に調べていき，中間変数を削除するなどして，状態数の減少を図ったところ，51 まで減らすことができた．

しかし，まだ二分決定グラフのノード数が大きくなり過ぎるため，次に二分決定グラフの変数順の最適化を人手で行っていった*．組合せ回路に対しては，二分決定グラフのための良い変数順を自動的に見つける発見的手法がいくつか提案されているが，SMV のような順序回路に対しては，あまり効果的な手法は見つかっていない．そこで，とにかく試行錯誤で，変数順を人手で変えながら，二分決定グラフが小さくならないかみていった．2,3 時間の試行錯誤の結果，二分決定グラフのノード数がかなり少なくて済む変数順が見つかり，それにより，SMV で検証できるようになった．

与えた仕様は，フィールドテストで見つかった異常動作が起きないこと，つまり，「同じアドレスが WAF 内に存在することはない」というものである．検証の結果，同じアドレスが WAF 内に存在する場合があり，その例が反例として表示された．その反例は 50 サイクルにもなっていた**．検証時間はワークステーションで8分程度である．

さて，反例を解析したところ，異常動作は以下のようにして生じることが分かった．empty-reset が1になると，WAF と RAF はリセットされる．しかし，リセットされる直前に RAF を出たアドレスはまだ RC の中にあり，WAF まで到達していない．これらのアドレスは WAF のリセットが終わってから WAF に到着するため，そのまま WAF に格納される．しかし，全てのアドレスが SC により，新しく順に作成されているため，結果的に同じアドレスが2個 WAF 内に存在するようになってしまう．つまり，empty-reset が1になったときに，RAF と WAF をリセットしたつもりが，実は RC に一部アドレスが残る可能性があるため，動作がおかしくなることが分かった．Xはこの結果を設計者に説明し，設計者も納得したところで，Xの仕事は成功のうちに終了したのである．ここまでくるのに，ABC の動作の理解，SMV の使い方の理解など含め

* アドレスデータがまったくない状態も識別しなければならず，アドレスを2種類用意するには，最低3種類，つまり最低2ビット必要となる．

* 二分決定グラフは一般に使用する変数順によってそのノード数が大きく異なることが多い．詳細については，本号の特集[4]を参照されたい．

** SMV はサイクル数が最小の反例を表示することになっており，その意味でこの反例をシミュレーションで見つけるのは大変であろうと予測される．

表-1 各モデルの大きさと検証時間

モデル名	状態変数の数	二分決定グラフの大きさ	検証時間
WAF	22	9,618	6 秒
State	23	16,791	69 秒
Data-path	51	47,202	468 秒

て，結局約3カ月かかった．

以上の各モデルの状態変数の数と二分決定グラフの大きさ，それに検証時間をまとめると**表-1**のようになる．使用したワークステーションはSUN SPARC2 である．

5. 形式的検証手法を利用する上でのポイント

Xの経験から以下のことが言える．

1. 回路からのモデル抽出は重要である．人手でやるのは本当に大変．VHDL などの標準のハードウェア記述言語からの自動変換は必須であろう．しかし，次のことも言える．

2. 状態変数が30個くらいから，すでに SMV の限界に近づいていると感じる．並列かつ独立に動作する複数のモジュールがあると簡単に二分決定グラフのノード数が爆発してしまう．したがって，回路を部分的に検証するか，設計を抽象化・簡略化しなければならない．この過程は現状では基本的に人手であり，結局は SMV の入力は人手で書くことになる．

3. CTL で仕様を表現するには，それ相当の熟練がいる．Xも始めは，仕様を正しく書けない（つまり，思っている動作と異なる仕様を書いている）ため，何を検証しているか分からない状態が続いた．しかし，1カ月もすると，ある程度自由に仕様を書けるようになった．一方，設計者が SMV などの検証ツールを直接使うとすると，このような教育期間を実際問題としてとれるかどうか，疑問が残る．設計工程上，時間があるときに，設計者を積極的に教育するようにしないと，検証は専門家に任せるという形でしか，使われることはないであろう（これが現状である）．

4. 形式的検証ツールは，反例を出してくれはするが，決して分かりやすくない．設計者の意図していなかったような場合であるから，設計者も間違ったのであり，反例の意味を理解するのも簡単とは言えない．これをなんとか分かりやすくする研究は急務かもしれない．

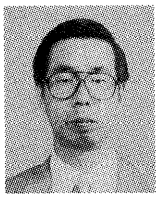
6. あ と が き

設計者には形式的検証ツールを勉強する暇がないのが現状であり，ここをなんとかしない限り，検証専用の人材を別途教育するというようなことをしなければならなくなる．つまり，形式的検証ツールが実設計で使われるようになるかは，形式的検証ツールが使えると言われたときに，上級管理職がどこまで本気でそれを信用するかにもかかっている．一方信用させるには，形式的検証ツールを適用した成功例を積んでいく必要がある．本稿で述べた経験は一つの成功の例とも言えるが，Xははっきり言って，大変だった．実チップのバグとりのように，せっぱ詰まらないととても最後までやろうと思わなかったかもしれない．その意味では，与えられた設計をきれいに検証するということよりも，具体的なバグをいかにして退治するかのような状況のほうが成功するかもしれない．つまり，形式的検証ツールは，従来言われていた本来の使い方とは異なる角度から，現場に導入されるような気がしている．

参 考 文 献

1) Bryant, R.E.: Graph-Based Algorithms for Boolean Function Manipulation, *IEEE Trans. Computer*, Vol. C-35, No. 8, pp. 677-691 (Aug. 1986).
2) Burch, J.R., Clarke, E.M., McMillan, K.L. and Dill, D.L.: Sequential Circuit Verification Using Symbolic Model Checking, In *Proc. 27th ACM/IEEE Design Automation Conf.*, pp. 46-51 (June 1990).
3) McMillan, K.L.: Symbolic Model Checking: An Approach to the State Explosion Problem, Technical Report CMU-CS-92-131, Carnegie Mellon University (May 1992).
4) 藤田，佐藤（編）：BDD（二分決定グラフ），情報処理，Vol. 34, No. 5, pp. 584-630 (May 1993).
5) 平石，浜口：論理関数処理に基づく形式的検証手法，情報処理，Vol. 35, No. 8, pp. 710-718 (Aug. 1994).

（平成6年1月5日受付）

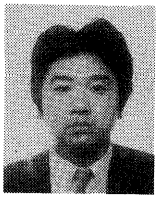


藤田 昌宏（正会員）

1956年生．1980年東京大学工学部電気工学科卒業．1985年同大学院工学系研究科情報工学博士課程修了．工学博士．同年富士通研究所入社．以来，ハードウェア設計用CADシステムの研究・開発に従事．1988〜1989年イリノイ大学計算機科学科客員研究員．1993年末より米国富士通研究所勤務．平成4年度情報処理学会研究賞受賞．IEEE会員．

陳 奔

1983年中国天津大学電子工学科卒業．1987年東京工業大学大学院電気・電子工学専攻修士課程修了．1990年同大学院同専攻博士課程修了．工学博士．同年東京工業大学工学部基礎集積工学講座助手．1992年富士通ディジタル・テクノロジ(株)入社．現在富士通(株)にて形式的検証の応用問題等に従事．回路シミュレーション，ASIC設計方法論，論理設計用CADに興味を持っている．電子情報通信学会会員．



山崎 正実（正会員）

1982年慶應義塾大学工学部計測工学科卒業．同年富士通(株)入社．論理合成システムなどASICの高位レベル設計技術の開発，及び通信機器向けASICの開発に従事．通信事業推進本部共通回路技術部に所属．