bluespec

あなたはESL設計基盤を砂の上に構築しますか？

「動作合成はESLの将来において‘非常に危機的’である、とCadence Design Systemsの合成ソリューションのcorporate vice presidentであるChi-Ping Hsu氏は語った。彼はさらに言及した。これまでのところ品質に問題があり、様々な種類のアプリケーションを扱う能力が制限されており、フォーマルベリフィケーションのメカニズムを欠いている」
EEDesign ESLデザインのための動作合成の評価、Richard Goering 2004年10月5日

全てのESLデザインフローの基盤は動作合成で、これはRTLを手作業で書き直しすることなく、高位のLSIデザインとモデリング作業を統合する。Gartner Dataquest Inc.,のchief EDA analyst のGary Smithは量産のLSIに使用できる動作合成ソリューションの必要性を強調した:「[動作合成ツール]なしにESLデザインフローをどうやって構築できるのか、私は分かりません。」
EEDesign ESLデザインのための動作合成の評価、Richard Goering 2004年10月5日

今まで、C/C++/SystemCのようなシーケンシャルなプログラムベースの動作合成ソリューションしか、RTLより高位のレベルには選択肢がありませんでした。これらのアプローチはデザインの抽象度を高めますが、効率的に適用できる狭い領域を除いて貧弱な合成品質などの重大な制限があります。その結果、ニッチなアプリケーションを除いてC/C++/SystemCは主に機能のモデリング、パフォーマンス調査、ベリフィケーションにのみ使用されてきました。実際のLSIを製造するためにRTLを書かないチップ開発チームはごくわずかです。

自分自身に問いかけてください:画像処理やフィルタなどの数学アルゴリズムを含まない動作合成のベンチマークを最後に見たのはいつか、ということを。おそらく、見たことがないでしょう - その理由はこれらのソリューションが抽象度は上げても、これら高位の構造を合成することは難問であるということに起因します。

Bluespecは、全てのアプリケーションでハンドコードのRTLに比肩する品質をもたらす高位合成の根本的に新しいアプローチを提供することによって、ハードウェアデザインを再発明しています。

ここでは、以下の側面に焦点をあててSystemCとBluespecを比較します：

- 構造
- リソース
- 並列性/連携
- コミュニケーション

ESLの考え方：RTLより高位にデザインを底上げする

ESLは、ハードウェアとソフトウェア両方を含むチップの完成品の生産性と品質を高めるというゴールに直接辿り着くことのできる、RTLより高位の設計のことです。工数にかかる費用やタイムツーマーケット、品質などの観点から、チップの複雑度はベリフィケーションやタイミングクロージャを行うチームが管理できる能力を超えてきたため、重要視される点が2つあります。

- ソフトウェアとファームウェアの開発を加速する
- タイムツーマーケットや工数を削減し、チップ開発の品質の問題のためにRTL設計を底上げする

ESLアプリケーションをターゲットとし、様々なカテゴリをカバーするEDAツールは多数あり、Gartner Dataquest's 2003 ESL Landscapeの一部だけでも1ダース以上あります。効果的なESLソリューションは以下の点を確実に実行する必要があります。

- 既存のEDAツールや設計手法と組み合わせて効率を上げる
- 仕様決め、モデリング、インプリメントの作業を統合
- チップ面積、周波数、レイテンシ、消費電力などの点でハンドコードのRTLの品質を保つ
- サブコンポーネント単体でなくプロジェクト全体を最適化する
- デザインの見通しや管理しやすさ、透明性を保つ

ESL合成：基盤となるもの

動作合成はESLの基盤を提供します。これが無いとモデリングの作業はハードウェアインプリメントから切り離されます。RTLのレベルで手作業によって書き直す作業は工数の無駄遣いで、モデルとの不一致やエラーを引き起こします。

Bluespec以前は、動作合成ソリューションはソフトウェアプログラミング言語環境、主にC、C++、SystemCがベースになっていました。C/C++合成は制御データフローグラフ(CDFG)を使用した、シリアルな記述からパラレルな記述へのマッピングを必要とします。SystemCはC++にパラレルなハードウェア構造を追加し、シーケンシャル・パラレルのハイブリッド環境を提供します。

SystemCは純粋なソフトウェアベースのハードウェア設計環境の制限に対応するために、C/C++にパラレル動作の構造を拡張したということについて、この資料でフォーカスします。加えてC、C++合成についても言及します。

SystemC、Bluespecと2つの異なる抽象度：根本的に異なるQoR

RTLのレベルを使っている場合、SystemCは効果的にVerilogに変換されます。しかしながら、抽象度が高いところでデザインが完了した場合には何が起こるでしょうか？SystemCとBluespecがどのようにデザインを高めているのかを構造、リソース、並列性/連携、コミュニケーションの4つの観点から見てみましょう。この解析はSystemCが抽象度のためにシーケンシャルな記述に依存していることが明確になります。これが原因で効率的な合成が高位デザインのベクトル化や、VLIWマッピング可能な数学アルゴリズムだけに制限されます。

次の表はSystemCとBluespecがデザインの抽象度を上げるために用いた方法についてまとめています。

Bluespec	明確 ＜LOC	明確 ＜LOC	ルールとメソッド ＜LOC	
SystemC	推測される構造とリソースを伴うシーケンシャルなアルゴリズム ＜LOC		明示的に管理	ワイヤ ＜LOC
RTL	明確	明確	明示的に管理	ワイヤ
	構造	リソース	並列性 / 協調	コミュニケーション

＜LOCはRTLよりコード行数が少ないことを示す

構造

SystemCの高位の設計生産性は、記述されたデザインの構造に依存します。そのパラレルなハードウェア構造によって、SystemCはVHDLやVerilogの設計抽象度に似たRTLのレベルとなります。実際のところ、設計者はVerilogやVHDLへの厳密な変換のためにRTLのレベルで設計してきました。しかし、これでは期間短縮もRTLより抽象度を上げることも実現できません。なぜ彼らはこれをやりたいのでしょうか？本来SystemCは、設計者が詳細なRTLマッピングをやめて、主なハードウェア構造の粒度や並列性を無視し、所望の関数処理を記述したシーケンシャルなアルゴリズム記述のトランザクションレベルで簡潔に書くときに、記述における有効性を発揮します。

SystemCとBluespec SystemVerilogの両方とも抽象度をトランザクションレベルで設計し、合成可能なインプリメントまでリファインすることができます。しかしながら、SystemCにおいてはこのリファインは、しばしば以下の問題があります。

- 計算モデルの変更（メッセージ通過モデルから RTL 信号化）はインタフェースの変更を伴う
- 合成可能になるのは低いレベル - RTL のレベルの SystemC

SystemCでは開発効率は高位のデザインによってもたらされます。この高位のデザインとはハードウェア構造を無視し、パラレルな構造をラッピングした、大部分はシーケンシャルなソフトウェアコーディングのことです。この動作合成を実行するための基本的なテクノロジを考察するとき、このタイプの抽象度はモデリングには有益ですが、合成ツールに2つの課題を課すとき、合成はひどいものになります。

- ツールがインプリメントを制御する場合、抽象物のアーキテクチャとマイクロアーキテクチャ、シーケンシャルアルゴリズムのレイテンシを決定する必要がある
- 加えて、ツールは簡潔なトランザクション表現を、効果的な、粒度の細かいインプリメントに変換する必要がある

SystemCベースの高位合成は抽象度、高位のシーケンシャルなコード、シリアル - パラレル変換テクノロジに依存し、以下の制限があります。

- C 言語の仕様はシーケンシャルであり、ハードウェアは本質的にパラレルである。パラレルプログラミングの分野の研究者は、シーケンシャルなプログラム(C/C++)をパラレルに変換するという表面的で魅惑的な目標をほとんど捨て去った。その研究はベクトル化や自動パラレル化の分野で 35 年以上前に始まり、パラレルなインプリメントのためにはシーケンシャルなプログラムは根本的に不十分な仕様であることが判明した。シーケンシャルなプログラムは、アルゴリズム的には不必要なシーケンシャル化を行うことで、コンパイラ/合成ツールがこのシーケンシャル化を元に戻すことは一般的に不可能である。ハードウェア合成というターゲットが異なるとしても(ベクトルマシンの代わりにハードウェア)、解析のテクニックは基本的に同じ問題を抱えている。
  シリアルなデザインにおける依存性の欠如が一般的に手におえないことが証明されるのと同時に、ごく限られた範囲の問題、単純な記述、単純なネストされた for ループ、線形のインデックスにおいてはパラレル化が効率的であることも分かった。これらは簡単にベクトル化でき、VLIW エンジンに直接マッピングできるアルゴリズムである。これはどういう意味であろうか？基本的に、フィルタやビタビデコーダのようなこれらの特性の数学アルゴリズムは、シーケンシャルなソフトウェアから効率的なハードウェアのインプリメントに変換できる
- C/C++をベースとするソフトウェア合成は DSP アルゴリズムの狭いマーケットにおいてのみ許容できる結果をもたらす。他の適用事例では C/C++はモデリングのみに格下げされる。

残念ながら、SystemCはC++をベースとするシーケンシャルなソフトウェアプログラミング言語です。そのため一般的なC/C++ベースのアプローチと同様の制限に悩まされるでしょう。

ネットワークルータのための以下のIPルックアップアルゴリズムの高位の記述を参照しましょう。これはパケットのIPアドレスから、フォワードすべき出力ポートを参照します。この単純なCベースのインプリメントでは、アドレス毎に1から3までのメモリテーブルを参照するために、ルックアップはまばらなツリー構造になります。

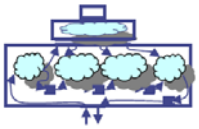
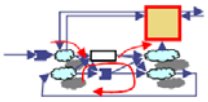
```
int longest_prefix_match (IPA ipa) /* Up to 3 memory lookups */
{
int p;

p = RAM [ipa[31:16]]; /* Level 1: 16 bits */
      if (isLeaf(p)) return p;

p = RAM [p + ipa [15:8]]; /* Level 2: 8 bits */
      if (isLeaf(p)) return p;

p = RAM [p + ipa [7:0]]; /* Level 3: 8 bits */
      return p; /* must be a leaf */
}
```

このシンプルなC言語記述をどのようにしてハードウェアにマッピングしますか？インプリメントで何サイクルかかるべきでしょうか？様々なメモリレイテンシをサポートするために、どのようなアーキテクチャが最も拡張性がありますか？いくつのルックアップをパラレル化できますか？これら全ての疑問は、ソースコードの中に指定されていないため、Cベースの合成ツールが解決すべきものです。以下の図は3つのアーキテクチャ例を示しており、それぞれ大きく異なる特性をもっています。ツールはどのようにして上述のシリアルな記述からこれらのアーキテクチャを決めるでしょうか？誰がハードウェアアーキテクチャを制御したいのでしょうか、ツールでしょうか、設計者でしょうか？

静的なパイプライン	線形のパイプライン	サイクル的なパイプライン
		
メモリ消費が非効率だが、シンプルなデザイン	メモリポートリプリケータを介した効率的なメモリ消費	効率的なメモリ消費で最も複雑な制御

Bluespecでは、設計者がアーキテクチャとマイクロアーキテクチャの両方のハードウェア構造を100%コントロールします。Bluespecの高位の抽象度は、RTLで実現できるインプリメントに比べて開発を迅速化し、エラーを削減することにフォーカスしています。

SystemCとBluespec SystemVerilogの両方とも表現力を向上し、構造体やユニオン、列挙型のような簡潔な高位の型をもっています。高位の型によって設計者は親しみのあるデータオブジェクトを使うことができ、コードの行数とエラーを削減できます。

残念ながらSystemCの力の源泉であるこれらの強みは、アキレス腱でもあります。あなたは高い抽象度と効率的な合成のどちらを望みますか？数学アルゴリズムを除いて、SystemCは両方を同時に満たすことはできません。

リソース

構造を構築するとき設計者は通常、ステートエレメントと、シフトや加算乗算などの演算子のような2種類のブロックを用います。これらがリソースで、ステートエレメントはSystemCがRTLから抽象化した領域です。ハードウェアのステートエレメントを明示的に記述したり、加算器やバレルシフタ、SRT除算器のように特定の処理ブロックを指定する代わりに、C/C++の表現力を合成に任せることができます。これにはいくつかのインプリメントに対する問題があります。

第一に、ソフトウェアからハードウェアへの変換のトレードオフを検討するために、SystemC動作合成ツールはテクノロジライブラリを必要とします。これらのライブラリは、様々なインプリメントがチップ面積や速度によって特徴付けられています。これによって合成ツールは様々なリソースの設定やリソース共有の影響を調査することができます。残念ながら、これらのライブラリによって、ASICやFPGAベンダの選択肢を事前に制限することになるでしょう。

第二に、構造に関するテーマを続けると、パイプラインレジスタのステージやステートエレメント、演算子のインプリメントや設定を決定するのに、合成ツールは本当に最善のツールなのでしょうか？2004年11月8日のEE Timesに掲載された “Getting an Algorithm Ready for Reuse”という題名の記事でKetul PatelはC++アルゴリズムを様々なプロセッサプラットフォームにポーティングする挑戦を記述しました。彼は「みんなはC++で記述されたアルゴリズムはプラットフォームを変更することが容易である、と仮定している。実際にはこれは正しくない。」と書いています。もしソフトウェアのコンパイラが同一のC++アルゴリズムを様々なプロセッサに効率的に最適化できないのであれば、どのようにしてハードウェア合成ツールが正しいアーキテクチャやマイクロアーキテクチャのインプリメントの選択ができるのでしょうか？彼は、画像処理アルゴリズムがターゲットのハードウェア TI TMS320C6205DSPにリコンパイルされたとき、要求よりも50倍遅かったと記述しています。アルゴリズムを有効にするために、大きな変更を加えました。

- C++からCに変換
- DMAコントロールを効率的に使用するように変更
- 浮動小数点から固定小数点への変更
- 固定少数点のインプリメントから除算を削除

最終のインプリメントでも事前の設定なしでは、コンパイラはターゲットのパフォーマンスに近づくことができませんでした。

第三に、合成ツールがリソースと構造のトレードオフを行うと、RTLはアルゴリズム開発者にとって理解できないものになるでしょう。合成ツールから出力されたRTLを使用することは、相当の努力を強います。唯一の選択肢はシミュレーションやデバッグにRTLを使用しないことです。RTLを避けられる可能性があるでしょうか？

最後に、消費電力のトレードオフや複雑なクロック構造、マルチクロックドメイン、IPの統合のようなその他の問題についてはどうでしょうか？これらはいつ、どうやって制御されるのでしょうか？

Bluespecでは、リソースの決定は明示的に行います。設計者がリソースを決定することが最善の方法です。これによって設計者はアーキテクチャやマイクロアーキテクチャを制御でき、結果が合理的で分かりやすいものになります。リソースを高位で表現できるようにするために、Bluespecは高位の型などのメカニズムを提供し、少ないコード行数で創造的なリソースのインプリメントを実現できます。高位の抽象度ながら100%設計者が制御できるのです。

並列性/連携

SystemCは構造とリソースの領域においてRTLより上位に抽象度を上げましたが、並列性やFSM間の連携の管理には何も新しいものがありません。シーケンシャルなプログラミング言語に並列構造できるように、SystemCに並列性のセマンティックが特別に追加されました。

しかしRTLのように、SystemC設計者は明示的に記述して並列性を管理する必要があります。何ら新しいメカニズムは追加されませんでした。事実、SystemCのセマンティックはVerilogやVHDLと比較しても簡潔ではありません。

対照的にBluespecでは並列性と連携に関してルールとメソッドの動作記述の新しいメカニズムによって革命を起こします。ルールはモジュール内でステートの動作をアップデートします。これは真のときアクションのかたまりが実行される、というように条件を記述します(下記の例を参照してください)。メソッドはインタフェースのプロトコルの動作でモジュール間のルールの動作を組み立てます。

ルールの構文:

```
rule rule_name (list of conditions);
    action_a;
    action_b;
    …;
    action_n;
endrule;
```

ルールの例:

```
rule stage1_leaf (unpack (sramResp.first) matches
  tagged leaf (.v));
      sramResp.deq;
      let {ip.lutag, itag} = fifo.first;
      fifo.deq;
      completionBuffer.complete.put(tuple2(itag,tuple2(unpack({0,v}),
         lutag)));
endrule;
```

ルールは、安全に連動するスマートなalwaysブロックであると考えることができます。安全に連動する、というのはRTLやSystemCとは異なり、Bluespecは以下を管理することです。

- 自動的に潜在的な競合条件を認識し、回避する
- 自動的にリソースの競合を認識する
- 多重化や接続性、必要なら共有リソースの調停を管理する
- 複数のアクションの同時実行などを保障する

Bluespecの基盤をなす項書き換え系(Term Rewriting System : TRS)テクノロジによって、これらの合成能力は設計者にとって簡単で透明性の高いものになります。しかしながら、これらのコンパイラの結果は設計者によって明確に指示され、簡単に制御できます。

対照的にRTLとSystemCではこれらの領域は手動で調整、管理され明示的に記述されます。

SystemCでは共有データメンバへの連携したアクセスには2つのメカニズムがあります。明示的にマルチプレクサや調停ロジックを設計することができますが、手動で詳細な作業を要します。代わりにモデリングの目的で共有データへの連携アクセスのためのみ排他制御やロックを使用することができますが、どのようなハードウェアにマッピングされるか明確ではありません。

デザインの複雑度が増大し、並列性や連携が膨大なエラーの原因となり、それらはしばしば分かりづらく、シミュレーションが困難で、見つけるのが困難です。SystemCではこれらのエラーを食い止めるためのRTL以上の機能はありません。

ルールとメソッドでは、Bluespecは並列性とFSM間の連携の仕様と管理の両方を底上げします。最も多く、分かりづらいバグと複雑化の原因が何であるか、自分に問いかけてみてください。構造とリソースもしくは並列性と連携ではありませんか？

コミュニケーション

デザインはサイズが巨大化し、ソリューションは多数のコンポーネントやコンポーネントの階層から組み合わされています。しかしながら、デザインを拡張するためにはRTLのテクニックは無力です。今日、インタフェースはとても単純化されています。

- ワイヤの定義を含む
- モジュール間のコミュニケーションのためのプロトコルは、通常、インタフェース毎にカスタマイズされておりモジュールのコアロジックの一部となっている
- プロトコルは、例えばタイミングダイアグラムやテキストなどのように非公式な形で文書化されており、モジュールを使用するたびに解釈し、明示的にコードに記載する必要がある
- モジュールはフックによって隣のモジュールと接続され、適切に相互作用するインタフェースのまたぐプロトコルが保障されない

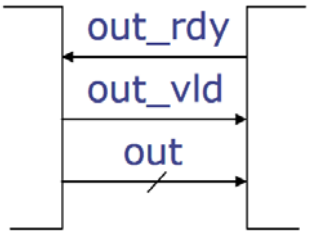
SystemCは同様のモジュール間コミュニケーションの古いモデルを使用しています。これは簡潔さの点においていくつかの利点がありますが、能力や問題点はRTLと同等です。

Bluespecは拡張性とリユースを促進するコンポーネントを組み立てるための新しいテクニックを組み込んでいます。

- インタフェースがプロトコルを含み、より簡潔な“信号”の定義を追加
- インタフェースメソッドは、手作業のコーディングや設定、余分なポートなどを必要とせず、他のモジュールからステートを変更することができる
- インタフェースプロトコルはモジュール内部のステートと弾性的に結合しており、インタフェースと外部モジュールのコアの処理レイテンシの変更に影響をうけない。例えば関数が 1 サイクル早く結果を出しても、デザインの変更が必要ない
- インタフェースはアサーションすなわち、それらが使われるか使われない条件を内包している。設計者は仕様書を読むことに足止めされず、ベリフィケーションチームはインタフェースが適切に設計されているかどうかを心配する必要がない

下記はSystemCコードの抜粋で、これらの機能を手動でインプリメントしています。このコードの抜粋では2つの例のインタフェースを接続するトップレベルモジュールがありません。Bluespec の下記の例では、example_interface_A は example_interface_Bを直接インスタンス化しており、それらを接続するためのトップレベルモジュールは必要ありません。

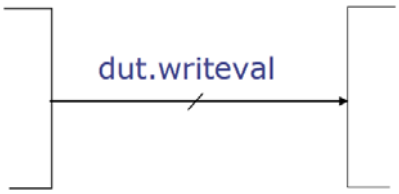
SystemCコードの例




```
SC_MODULE(mod_A) { ...
sc_out<SC_unit<32>> out;
sc_in<SC_unit<bool>> out_rdy;
sc_out<SC_unit<bool>>
out_vld;
...   };
void
mod_A::entry() {
   if (!rst) {
      out.write(0);
      out_vld = 0;
      wait(1);   }
while (true) { ...
   y = f(...);
   do {
      wait(1);
   } while (!out_rdy);
   out_vld = 1;
   out.write( y );
      wait(1);
      out_vld = 0;
   } }
```

```
SC_MODULE(mod_B) { ...
sc_out<SC_unit<32>> out;
sc_out<SC_unit<32>> out_rdy;
sc_in<SC_unit<32>> out_vld;
... };
void
mod_B::entry() {
   if (!rst) {
   out_rdy = 0;
   wait(1);
   }
while (true) {
   ...
   out_rdy = 1;
   do {
      wait(1);
   } while (!out_vld);
   x = out.read();
   out_rdy = 0;
 ...
 } }
```

Bluespec SystemVerilogコードの例




```
module mod_A ();
...
intfc_B dut();
mod_B the_dut (dut);
rule foo_bar;
      ...
      y <= f(...)
      dut.writeval (y);
      ...
endrule
...
endmodule
```

```
interface intfc_B;
      method Action writeval
        (bit[31:0] out);
endinterface;

module mod_B (intfc_B)
   method Action writeval(out) if
      (mod_rdy);
      x <= out;
   endmethod
endmodule
```

これらの2つのコードの例は、単純なシナリオを表しています。mod_Aとmod_A'の2つのモジュールがmod_Bに対して同じやり取りをするようにわずかな変更を加えることをイメージしてください。Bluespecではツールがリソースの競合を認識し、適切な接続や管理するための制御ロジックを生成するため、たいしたことではありません。SystemCでは、このデザインへの小さな追加仕様が、接続性や多重化、調停制御などの無数の詳細な変更を必要とします。

Bluespecのコミュニケーションにおける能力は、迅速な組み立てと設定の変更の両方に寄与します。デザインのリユースは、出来合いのIP以上のものを意味し、設計者の労力を削減し、インタフェースのための複雑な制御ロジック構築を削減することを意味します。インタフェースは使用に際して自己文書化され、迅速な接続性と使用時のエラーを大幅に削減します。

結論

デザインのレベルをRTLより上に上げるためには、いくつかのアプローチがあります。様々なアプローチはデザインを大幅に迅速化すると共に、それらがデザインのレベルを上げる方法は合成とデザインのエラーを減少させる際に非常に異なった意味をもっています。

性能とタイムツーマーケットを改善する材料を提供するために、ESLソリューションは以下の要件が必要です。

- RTL デザインの全ての側面を高め、加速する。一分野(数学アルゴリズム)のみを加速し、良い品質にすることは、ターゲットのリソースを改善し、柔軟性を増すことになるが、SoCプロジェクトにおいては、RTL の大半の開発とベリフィケーションは改善されないという弱点によって制限される。ESL は全ての問題を解決すべきで、ゴールはシステム開発の一部分だけでなく、システム全体を加速することである
- RTL のレベルのツール、ベリフィケーションと統合されていること。ESL ツールは以下の点を無視できない
  - 既存の IP
  - 十分に理解され、構築されている既存の EDA メソドロジ
  - 高位のデザインと RTL を比較する等価性検証ツールが存在しないこと

設計チームはベリフィケーションとデバッグをRTLのレベルで行う必要があり、それはこれから何年も変わらないでしょう。ESLツールは実用的である必要があり、高位のみではなくRTLなどの設計全体の生産性を上げる必要があります。

もしあなたがSystemCベースの動作合成ツールを探しているのなら、下記のようなデザインにおいて、チップ面積、速度、レイテンシ、コードの行数をRTLとSystemCを比較したベンチマークを示すように、ベンダに依頼してみてください。

- プロセッサ
- バス調停回路
- ネットワーク処理
- 複雑な制御ロジックのアプリケーション
- ベクトル化や VLIW エンジンへのマッピングが容易でないアプリケーション

強力なモデリング環境のみを必要としているなら、それも結構です。もしゴールがハードウェアへのインプリメントであれば、一般的にモデリングとハードウェアインプリメントのための高位の環境は、好ましくはありません。仕様決めとインプリメントの片方が抜けているなら、作業は分断され、二度手間となるでしょう。

SystemCとは異なり、Bluespecは全てのアプリケーションにおいてハンドコードのRTLに比肩する品質を実現する、デザインの底上げによって、ハードウェア設計に再革命を起こします。

お問い合わせ先：

CYBERNET

サイバネットシステム株式会社
新事業統括部
〒101-0022 東京都千代田区神田練塀町３富士ソフトビル
Tel: 03-5297-3295　Fax: 03-5297-3637

e-mail: bluespec@cybernet.co.jp
http://www.cybernet.co.jp/bluespec/