Vol. 33 No. 11　　情　報　処　理　　Nov. 1992

解　説

ハードウェア記述言語

2. 主要なハードウェア記述言語の特徴と標準化状況

2.3 SFL†

小栗　清††　中村行宏††
野村　亮††　名古屋　彰††

1. まえがき

SFL (Structured Function description Language)[1)~4)] は記述対象を同期回路に限定し，すべての記述を手続きの中で行うように制限した新しいタイプのハードウェア記述言語である．

信号の接続と信号間の因果関係により対象を表現する言語が多い中で，接続の記述を排除して，手続き記述に徹したところに特徴がある．この単一パラダイムの採用により，従来一体となって行われていた LSI ハードウェアの動作設計と論理設計が分離され，論理設計が自動化された新しい設計環境が実現されることとなった．

2. 言語設計の基本思想

SFL は，すでに存在するハードウェアの設計様式を記述する言語を開発することではなく，従来の設計手法に比べ，効率的な設計手法を提供することを目的に開発された．

このため，言語，評価系，合成系，テスト生成系が，同時に開発され，記述性の向上や設計の自動化のための条件が明確になってから言語仕様が公開された[4)]．

2.1 手続き

SFL での最も特徴的な制約は，すべての記述が手続きの中で行われるという点である．

LSI 設計のうち，動作設計と論理設計は，動作設計の結果を表現する適切な形式がないために，渾然一体として行われており，その結果は，部品間の接続関係（回路図）として記述されることが多い．ところが，設計者は接続表現からいったん動作を抽出しなければ回路を理解できないので，接続表現による設計では，設計内容を接続に変換して記述し，記述から動作を抽出理解して，設計を追加あるいは完成させていくという負担の大きい作業が必要である．

この負担を除くために，動作メカニズムを明示的に表現できる言語を用意して，動作設計と論理設計を分離しようというのが SFL 開発の目標となっている．

一方，ハードウェア化の目的は主に並列動作による処理の高速化であって，手続き記述は並列動作の表現に不利なのではないかという疑問がある．

ところが設計者が並列度を増すように工夫するとき，それは人間社会（会社組織など）の並列性をモデルとすることが多い．

人間社会の並列性とは，どちらかというと一つのテーマを逐次的に進めるときに最大の成果をあげることができる個々人へ，作業の分担指示や移管により仕事が分散されることにより，組織全体としては，非常に高い並列性が得られるというような構造のことである．この様子を**図-1** に示した．

すなわち，並列動作と手続き的（逐次的）記述とは矛盾するものではないといえる．ただし，手続き間の制御は並列性の記述能力に大きな影響を与える．SFL では手続き間の制御のためにハードウェアタスクという概念が用意され，分担指示や移管といった概念がそのまま表現できる(3.)．

2.2 刺激反応モデルとの比較

ハードウェア内の事象の発生状況を信号の因果関係と信号の接続関係で記述する方法がある(**図-2**(a))．信号間の因果関係のうち，向きおよび値の関係を手続きにより，原因から結果までの経過時間を数値として表現することにより，事象が次々と生起される様子を間接的に表現することが

† SFL by Kiyoshi OGURI, Yukihiro NAKAMURA, Ryo NOMURA and Akira NAGOYA (NTT Communication Science Laboratories).

†† NTT コミュニケーション科学研究所

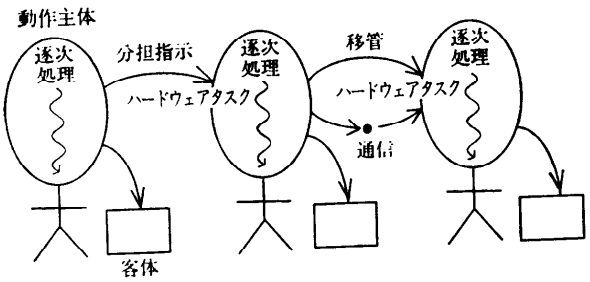
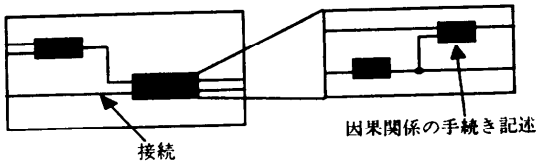
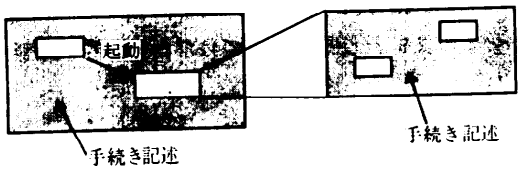


図-1 複数の手続きによる並列性の表現



(a) 刺激反応モデル



(b) 手続きモデル

図-2 刺激反応モデルと手続きモデル

できる．これは刺激反応モデルと呼ばれ，VHDLなどで採用されている考え方である．ただし，刺激反応モデルでは事象の推移を手続きとして表しているのではないことに注意する必要がある．また，刺激反応モデルでは，階層構造の最下位のみが手続き記述で，その上位は接続記述となることが多い．これに対し，SFL では，すべての階層が手続きとして表現できるように工夫されている(2.5)．

2.3 時間の抽象度・構成要素の抽象度・データの抽象度：自動化とのバランス

時間に関する抽象度のレベルには，実時間，マシンサイクル単位，順序があり，記憶機能や演算・加工機能の表現のレベルには，ハードウェア構成要素を明示しての表現，変数や演算子による表現がある．さらに，データの型に関しては，論理のみ，2進数による正整数のみ，整数，実数，複素数，あるいはそれらの複合型というように選択の幅は広い．これらの抽象度が SFL においてどうであるかをみていこう．SFL における抽象度の決定は主に設計自動化とのバランスにより行われている．

「どのようなデータをどのような順序でどのように加工するか」という手続きの記述から回路を合成するためには，

① どのデータをどのレジスタに割り付けるか，

② どの加工をどのような演算回路で行うか，

③ レジスタや演算回路をどのマシンサイクルで，またどの順序で動作させるか，

④ レジスタ，演算回路間のデータ転送路(セレクタ，バス)をどうするか，

⑤ 各演算回路をどう構成するか，

⑥ 順序をどのような状態遷移で表すか，状態をどのようなコードに割り付けるか，

⑦ 条件判定，回路の制御，状態遷移，論理式の評価などをどのような回路で行うか，

⑧ どのような部品で回路を構成するか，部品のもつ各種条件をいかに満足させるか，

を決定・最適化していかなければならない．①～⑧は

a．アーキテクチャレベルの設計，最適化

b．順序回路の設計，最適化

c．組合せ回路の設計，最適化

d．テクノロジマッピング

のいずれかに分類され，a～dはそれぞれ最適化手法が研究されている．ところが，演算回路を代表としてその設計がa～dのすべてに関係し，その分担のバランスが重要である対象は少なくない．

そこで，基本的な自動最適化（b，c，d）以外の最適化の結果，すなわち回路の構成や手続きの各マシンサイクルへの配置（b，c，dの分担のバランスならびにa）を明示的に記述できる言語を用意し，合成系を記述そのものの最適化を行うものと，記述に基づいて最適化を行うものに分けるのが望ましいと考えられる．SFL ではこのような考え方に従って，記述の抽象度は，時間については同期回路を前提とするマシンサイクル単位，構成要素についてはすべて明示とされた．

これは，これまでに蓄積されてきた多くの有用な構成法を計算機に発見させることはいずれにしても困難であるならば，結局それらの構成法を計算機に入力しておく必要があるわけで，それならばその表現を SFL という形式の中で統一的に扱うほうがシステムの開発者にとっても利用者にと

っても都合がよいはずであるという考えに基づいている.

この結果，演算回路のように回路の構成法がよく研究されている対象や最適構成が考慮された設計に対し，その SFL 記述から基本的な最適化のみを行うことにより高品質な合成結果を得ることができることとなった.

図-3 に，組合せ回路による乗算器の記述例を示す．小学校で教わった筆算による乗算の方法がほぼそのまま表現されている点に注目されたい.

SFL のデータの型は，冗長2進数による高速演算器などが，ますます重要となってくる傾向であるので，論理値と論理の束値だけとされた.

2.4 従来の RTL 言語からオブジェクト指向言語へ

従来の RTL 言語は，記述対象に含まれるレジスタ，メモリ，端子のみをハードウェア構成要素とし，組合せ回路に対しては演算子を使用するのが普通である．しかし，変数と演算子の組合せに比較すると，レジスタと演算子の組合せはバランスが悪い．というのは，動作の意味が「レジスタへの書き込み」だけに限定され，また，動作レベルの設計で重要な組合せ回路の共用化などが表現できないためである.

SFL では，「動作」は「構成要素の起動手順と起動にともなうデータの依存関係」と一般化されていて，レジスタと同様，組合せ回路も構成要素として動作記述の中で陽に扱われる．**図-4**(a)に構成要素の起動とデータの依存関係を示す SFL の典型的な記述例を示した．**同図**(b)はこれに対応する回路構造である．マルチプレクサは(a)以外のデータ依存関係によるもので，(a)が回路構造をそのまま表しているのではないことを示す例とした.

```
%i 〈csa16.h〉 ライブラリの引用
%i 〈cpa16.h〉 csa はキャリーセーブアダー
module multi8 {        モジュールの定義
    input  in1〈8〉 ;
    input  in2〈8〉 ;    インタフェースの定義
    output out〈16〉 ;
    csa16  csa0, csa1, csa2, csa3, csa4, csa5 ;
    cpa16  cpa ;        構成要素の定義
    tmp    s0〈16〉, s1〈16〉, s2〈16〉, s3〈16〉,
           s4〈16〉, s5〈16〉, s6〈16〉, s7〈16〉 ;
    tmp    t0〈16〉, t1〈16〉, t2〈16〉, t3〈16〉 ;
    tmp    u0〈16〉, u1〈16〉, u2〈16〉, u3〈16〉 ;
    tmp    v0〈16〉, v1〈16〉 ;
    instrin do ;
    instruct do par {    以下が手続きの記述
        /*·····*/
        s0=0b00000000 || (in1 & 8 # in2〈0〉)          ;
        s1=0b0000000 || (in1 & 8 # in2〈1〉) || 0b0 ;
        s2=0b000000 || (in1 & 8 # in2〈2〉) || 0b00 ;
        s3=0b00000 || (in1 & 8 # in2〈3〉) || 0b000 ;
        s4=0b0000 || (in1 & 8 # in2〈4〉) || 0b0000 ;
        s5=0b000 || (in1 & 8 # in2〈5〉) || 0b00000 ;
        s6=0b00 || (in1 & 8 # in2〈6〉) || 0b000000 ;
        s7=0b0 || (in1 & 8 # in2〈7〉) || 0b0000000 ;
        /*·····*/
        t0=csa0.do(s0, s1, s2).out1 ; t1=cas0.out2 ;
        t2=csa1.do(s3, s4, s5).out1 ; t3=cas1.out2 ;
        /*·····*/
        u0=csa2.do(t0, t1, t2).out1 ; u1=cas2.out2 ;
        u2=csa3.do(t3, s6, s7).out1 ; u3=cas3.out2 ;
        /*·····*/
        v0=csa4.do(u0, u1, u2).out1 ; v1=csa4.out2 ;
        /*·····*/
        csa5.do(v0, v1, u3) ;
        /*·····*/
        out=cpa.do(0b0.csa5.out1, csa5.out2).out ;
```

図-3 8×8 乗算器の SFL 記述

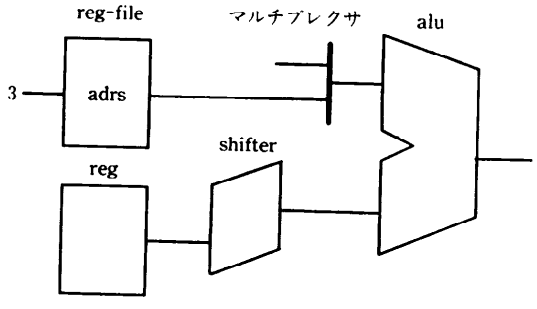
対象．指示（引数，引数…）．結果の形式は，「ある対象を特定の指示により起動して動作させ，結果を得る」ということを表しており，オブジェクト指向言語のメッセージパッシングと同等である．ただし，SFL では任意個の対象が同時に動作することになっても構わない.

2.5 動作記述に閉じたリカーシブ性

対象を構成要素間の接続で示すとき，その構成要素は，またさらに下位の構成要素間の接続で示されるというように，構成要素と接続に関し，接続記述はリカーシブな特性を有している．このため，接続記述は，自然に階層化や部品化が行え，

```
·····alu.add(reg-file.read(0×3).out,
             shifter.left(
               reg.read( ).out
             ).out
           ).sum·····
```

(a) SFL 記述



(b) 対応する構造

図-4 構成要素の起動手順とデータの依存関係の表現

また，工程によらず，PMS（プロセッサ，メモリ，スイッチ）のレベルからトランジスタレベルまで使用することができる．これが回路図による設計が広く行われている大きな理由である．

動作記述を行う場合も，ある特定のレベル，たとえばレジスタ・トランスファ・レベルだけが動作記述で，その上位は接続記述というのでは，階層設計の自由度が低く，全体としての効率を向上させることができない．

先に述べた刺激反応モデルでは，刺激反応モデルで記述された対象の外側は接続記述となってしまうので，このような問題をもつことが多い．一方，SFL では，前節で述べた「動作の一般化」により，動作を対象の起動手順としたために，動作記述の中で対象，すなわち階層を扱うことができ，接続記述がそうであったように，PMS レベルからゲートレベルまで，同じ形式で扱うことができる（**図-2**(b)）．

図-5 に示すように，メッセージは上位から下位構成要素へ，また下位構成要素から上位へ渡す必要があり，おのおの，instrin, instrout で定義される特別の端子（指示入力端子，指示出力端子）が用意されている．下位構成要素から上位へ渡されるメッセージに基づく動作は上位構成要素の記述であることに注意されたい．

2.6 設計の蓄積・ライブラリへの分離，ライブラリの引用法

SFL の抽象度が設計対象ごとの構成そのものを記述できるレベルであることと動作記述の中で物理的階層を自然に表現できることは，高品質な設計を蓄積し再利用していくための重要なポイントである．

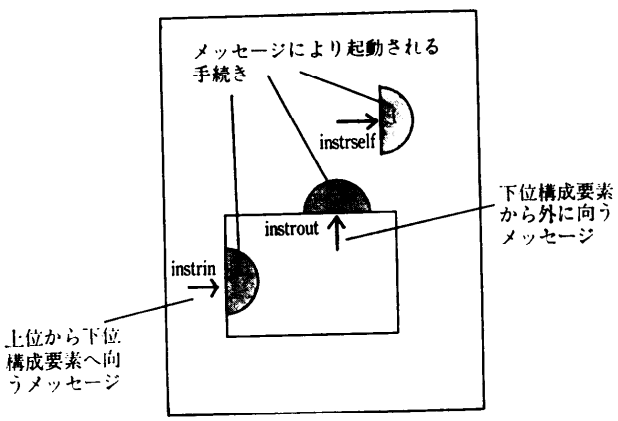


図-5 メッセージパッシングと階層構造

SFL には加算や乗算，あるいはシフトといった演算子は組み込まれていない．したがって，加算器や乗算器，バレルシフタなどは SFL によって一度は記述しておかなければならない．もちろん利用者は単にこの記述を利用すればよいので，演算子として組み込まれている場合と大差はない．

図-3 の %i 〈csa 16. h〉 はライブラリからのキャリーセーブアダーの引用を示している．c 言語でのヘッダファイルのインクルードと同様の感覚である．

2.7 タイミング設計の分離・自動化

従来，非同期回路が実際に論理のとおりに動作するかどうかはもちろんのこととして，同期回路による設計であっても，回路が同期回路として動作するかどうかは，設計者の責任であった．特に，設計のこの部分はタイミング設計と呼ばれ，多くの工数を要している．

同期回路のクロックに挟まれる時間内（1 周期）の電気的動作を次のように考えよう．以下は同期回路の説明として当たり前のことを述べているに過ぎないが，遅延の伝播がクロックパルスに始まり，レジスタの入力端子に終わるという認識は重要であるので少し詳しく述べた．

すべてのレジスタのクロック入力端子は外部クロック入力端子に接続されていて，1 周期に 1 個のクロックパルスが入力される．

すべての端子の電位の変化は，外部データ入力端子，外部クロック入力端子に始まり，次々と伝わる．レジスタ以外の素子は入力端子の変化を出力端子へ伝える．レジスタはクロック入力端子の変化を出力端子へ伝える．変化は外部出力端子またはレジスタのデータ入力端子まで達してそこで終わる．

素子は複雑に接続されているから，端子には種々の経路から伝播してきた変化が重なり，その波形は複雑なものとなる．しかし，レジスタ以外に記憶素子はなく，したがってループは存在しないので，十分に時間がたてば，すべての変化は伝播し終えて，回路内には変化が存在しない状態となる．この時点で次のサイクルのクロックが投入されるものとする．このクロックによってレジスタの

データ入力端子の値がレジスタに記憶されるとともに出力端子へ出力され，次のサイクルの変化が始まる．

このように同期回路では，タイミング設計は変化の伝播状況の計算と変化の終着点に存在する制約条件の評価とその充足に帰着させることができる．

SFL では，タイミング設計の自動化を可能とするために，クロック信号を直接操作することが禁止されている．

3. 言語の主な特徴

前章では，幾つかの観点から，SFL 言語がどのように設計されているかを述べた．ここでは，SFL 言語の構造に沿って，その概要を述べることとする．

SFL の記述は，物理的境界をもつ部品種（モジュール）を単位に行う．

実際の管理単位であるファイルには階層をなす複数のモジュールと，引用されるモジュールのテンプレートを並べる形で記述するのが普通である．モジュール名は部品種名そのものであり，全体で一意でなければならない．

モジュールの定義は，構成要素の定義，手続きの存在と仮引数の定義，手続きの内容の記述からなる．

2.5 に述べたように，手続きはモジュールへのメッセージパッシングを表す特別な端子（指示入力端子，指示出力端子）に対応づけられて記述される．実際には instrself で定義される指示内部端子による手続きがあり（図-5），手続きの共通部分をまとめるのに役立っている．

さらに，手続きには，起動されたマシンサイクルだけの動作である指示端子による手続きのほかに，一度起動されると複数マシンサイクルにわたって状態遷移しつつ動作しつづけるステージ（stage によって定義される）による手続きがある．少なくともステージの最初の起動は指示端子から行われることに注意したい．

手続きの本体は他の手続きの起動を示す形式の並びである．ただし，ステージでは状態遷移のための形式もある．

手続きの起動を示す形式には，

構成要素．指示入力端子（引数，引数，…）．結果

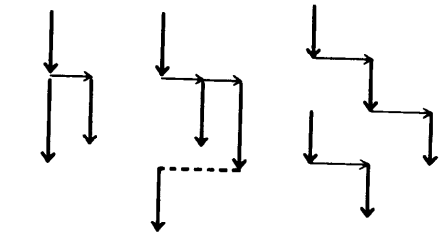


図-6 ハードウェアタスクによる並列性の表現

指示出力端子（引数，引数，…）．結果

指示内部端子（引数，引数，…）．結果

generate ステージ．ハードウェアタスク（引数 引数，…）

relay ステージ．ハードウェアタスク（引数，引数，…）

がある．relay を除き，起動のたびに処理の並列度が増す．relay では起動元の動作は終了する．ハードウェアタスクはステージに対するメッセージの扱いである．図-6 にハードウェアタスクによる並列動作の表現例を示した．

引数は，手続きを開始するために必要なデータであり，ここに上記の形式をネスティングして使うこともできる．データは定数とこの形式を演算子で連結したものである．また起動のみで結果を必要としない場合は「．結果」を除く．

演算子には

^：否定

|：論理和

@：排他論理和

&：論理積

/|：束方向の論理和

/@：束方向の排他論理和

/&：束方向の論理積

#：MSB の拡張

||：連結

〈n：m〉：切り出し

がある．

手続きの起動は

if (condition) 起動の形式

のようにして条件付けることができる．condition は 1 ビットのデータである．

4. 処理系の開発状況

処理系の開発は言語の設計と並行して行われ，動作シミュレータ seconds と合成系 sflexp/optmap

が利用可能な状態にある.

以下に各処理系の特徴を述べる. seconds は SFL で記述された動作の流れを直接解釈して実行する動作シミュレータである. ①前処理として接続論理への展開を行わないためただちにシミュレーションを開始できる, ②同一対象に対する多重書き込みエラーなどを検出できる, ③手続きの実行位置, 各データの値を詳細にトレースできる, などの特長をもっている.

sflexp は SFL 記述からテクノロジに独立な回路を生成するシンセサイザ, optmap は自立制約違反解決メカニズムによりテクノロジマッピングならびに負荷設計, タイミング設計を行うプログラムである.

sflexp/optmap では, おおよそ次のような処理が行われている(図-7). ①構成要素の起動手順と起動にともなうデータ参照, 加工, 転送から, データ転送構造を生成する, ②データ転送の性質に応じてドントケア条件を付加しつつ, おのおののデータ転送構造を制御するまたは構成する論理を簡単化する, ③すべてのデータ転送構造を合わせて全体の回路とする, ④実回路の規模によい相関をもつ仮想ゲートの総ファンイン数を最小化するように回路を多段化する, ⑤実部品に変換する, ⑥目標性能に達しない部分を必要ならば回路の規模を大きくして, 性能を改善する. ④, ⑥により, 目標性能を満たす最も小さな回路が得られると期待できる[5].

テストパターンは次のようにして生成されている. ①抽出されたデータ転送構造のみを使用して外部端子から任意の値を設定(可制御), あるいは外部端子へ値を転送(可観測)できるレジスタ群とそのための転送手順を決定する, ②この転送を本来の転送に加えて論理合成する, ③可制御可観測でないレジスタにスキャンパスを挿入する, ④最終的な回路図情報の組合せ回路部分に対してテストパターンを生成する, ⑤先の転送手順を実施するパターンと組み合わせて最終的なテストパターンとする.

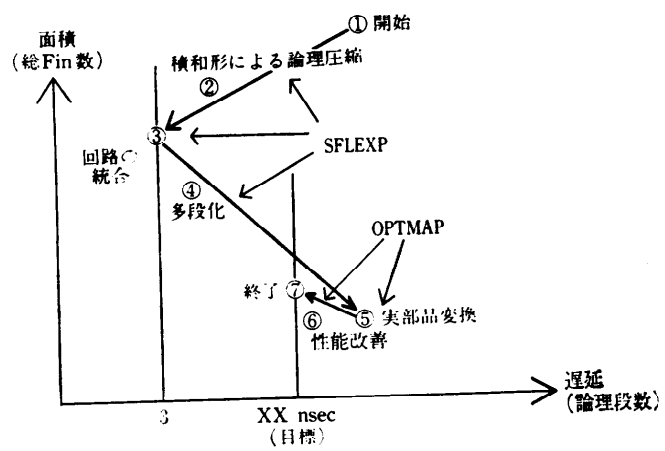


図-7 合成系の処理

今後の予定として, 分散独立クロックシステムへの対応, FPGA への対応, スケジューリング, リソースアロケーション, データ転送路の自動割付などのハイレベル最適化, 利用者によるアルゴリズムの追加を容易とする研究用プラットフォームとしての整備などが予定されている.

5. 標準化の状況

SFL は, 言語, 評価系, 合成系が単一の組織で同時に開発された唯一のものとしてその完結性やコンパクトさに特徴がある. 大規模かつ複雑な記述言語が開発される中で, 対象を同期回路に絞り, すべてを手続きとして記述させると, どのような設計環境が実現できるのかという見本となっている.

SFL 言語の組織的な標準化活動は行われていないが, 言語仕様は公開されているので, 上記の特徴を活かした利用が進むものと思われる.

相互利用ができる SFL ライブラリや使用ノウハウの蓄積のための組織作りが開始されているようである.

一方, SFL から VHDL への変換プログラムが検討されており, これは SFL を採用する場合のリスクを軽減することとなろう.

6. 応用例など

FDDP (Four-Day-Designed Processor) はデータパスからパイプラインの制御まですべてを SFL によって設計記述され, 自動論理合成, 自動レイアウトの後, 製造され, 動作が確認された 32 ビット RISC プロセッサである. 本プロセッサは MIPS R3000 によく似た DLX[6] のサブセット 47 命令を

実装し，ハーバードアーキテクチャ，5段パイプラインで構成されているにもかかわらず，SFLによる記述は，パイプラインのコントロールに665行，ALUやバレルシフタなどのデータパスに564行という非常に簡潔なものである．

このほかにも画像処理用DSP，符号処理LSIなどSFLによるLSIの設計，製造事例が報告されている[7]．

7. あとがき

動作記述言語SFLの設計思想，言語仕様，処理系ならびに適用例について述べた．

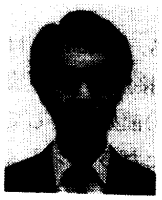
SFLは，単純な構成ながら，ALU，乗算器のような組合せ回路から，先行制御，置いてきぼり制御，パイプライン制御を酷使した高並列プロセッサまでを容易に記述することができる．

SFLは，言語，評価系，合成系が単一の組織で同時に開発された唯一のものとしてその完結性やコンパクトさに特徴がある．大規模かつ複雑な記述言語が開発される中で，SFLはその特徴を活かした利用が進むものと思われる．

参考文献

1) 中村行宏，小栗 清：ハードウェア記述言語とその応用，情報処理，Vol. 25, No. 10, pp. 1033-1040 (1984).
2) Nakamura, Y., Oguri, K., Nakanishi, H. and Nomura, R.: An RTL Behavioral Description Based Logic Design CAD System with Synthesis Capability, Proc. 7th International Conference on Computer Hardware Description Languages and their Applications (IFIP CHDL 85), pp. 64-78 (Aug. 1985).
3) Nakamura, Y.: An Integrated Logic Design Environment Based on Behavioral Description, IEEE Trans. CAD, CAD-6, 3, pp. 322-336 (May 1987).
4) 中村行宏，小栗 清，野村 亮：RTL動作記述言語SFL，電子情報通信学会論文誌A，Vol. J 72-A, No. 10, pp. 1579-1593 (1989).
5) 名古屋彰，中村行宏，小栗 清，野村 亮：高位記述からの大規模論理合成における多段論理最適化，電子情報通信学会論文誌 A，Vol. J 74-A, No. 2, pp. 206-217 (1991).
6) Hennessy, J. L. and Patterson, D. A.: Computer Architecture a Quantitative Approach, p. 594, Morgan Kaufmann Publishers, inc., California (1990).
7) 佐藤 淳，大槻 遇，服部圭佑：市販論理合成ソフトで2万5000ゲートのチップを設計，日経エレクトロニクス，No. 553, pp. 197-208 (1992).

(平成4年6月23日受付)



小栗 清（正会員）

昭和49年九州大学理学部物理学科卒業．昭和51年同大大学院修士課程修了．同年日本電信電話公社入社．同社電気通信研究所において，DIPSアーキテクチャ，論理装置などの研究，設計開発を経て，並列処理アーキテクチャ設計技術の研究に従事．現在，NTTコミュニケーション科学研究所主幹研究員．平成4年4月より電気通信大学大学院情報システム学研究科客員助教授．1987年第2回元岡賞受賞．平成元年度情報処理学会論文賞受賞，平成3年度大河内記念技術賞受賞．電子情報通信学会会員．

中村 行宏（正会員）

昭和42年京都大学工学部数理工学科卒業．昭和44年同大大学院修士課程修了．同年日本電信電話公社入社．同社電気通信研究所において，DIPSアーキテクチャ，論理装置などの研究開発を経て，並列処理アーキテクチャならびに高位設計方式の研究に従事．現在，NTTコミュニケーション科学研究所研究グループリーダ，主幹研究員．平成4年4月より電気通信大学大学院情報システム学研究科客員教授．平成元年度情報処理学会論文賞受賞，平成3年度大河内記念技術賞受賞．電子情報通信学会，IEEE各会員．

野村 亮（正会員）

昭和57年大阪大学基礎工学部情報工学科卒業．同年日本電信電話公社（現NTT）入社．同社電気通信研究所において，ハードウェアの設計支援，並列アーキテクチャの研究に従事．現在，同社コミュニケーション科学研究所主任研究員．平成3年度大河内記念技術賞受賞．

名古屋 彰（正会員）

昭和53年京都大学工学部電気系学科卒業．昭和55年同大大学院修士課程修了．同年日本電信電話公社（現NTT）入社．同社電気通信研究所において，DIPSアーキテクチャ，論理装置の研究開発などを経て，ハードウェア設計支援，並列処理アーキテクチャの研究に従事．平成2年より3年までイリノイ大学客員研究員．現在，同社コミュニケーション科学研究所主幹研究員．平成3年度大河内記念技術賞受賞．電子情報通信学会会員．