# ポストレイアウトシミュレーションのための SPF ファイル縮小化に関する一考察

米田　友洋[1,a)]

**概要**：レイアウト後にチップレベルの詳細な SPICE シミュレーションを行いたい場合があるが，通常，時間がかかりすぎる，あるいは，必要メモリ量が大きくなりすぎる，といった要因により容易には行えない．そこで，観測したい出力線群を指定し，その信号線の振る舞いのみに注目することで，シミュレーションを容易化することを考える．本報告では，寄生 RC 抽出により得られる SPF(Standard Parasitic Format) ファイルを対象とし，指定された出力線群に対し，そのシミュレーションには不必要な部分をその SPF ファイルから選択的に削除することで，高速かつ少ないメモリ量で，より正確なシミュレーションを行う手法について議論するとともに，初期実装による簡単な実験結果を示す．

# An Idea of SPF File Reduction for Post-Layout Simulations

Yoneda Tomohiro[1,a)]

***Abstract:*** It is not usually easy to perform precise post-layout chip-level SPICE simulation due to very long simulation time and large memory space required. One possible approach to practical post-layout simulations is to simplify the simulation model by focusing the behaviors of specified output signals. This report proposes an idea to reduce SPF (Standard Parasitic Format) files obtained by a parasitic RC extraction tool such that the preciseness of the simulation with respect to specified output signals is preserved, and shows preliminary experimental results obtained by a naive implementation of the proposed algorithm.

## 1. はじめに

LSI 試作，特にアナログ回路を含むミックスドシグナル LSI を試作する場合には，パッド等を含めたレイアウト済みデザインに対して詳細な SPICE シミュレーションを行いたいことがある．詳細なシミュレーションのために通常寄生 RC 抽出を行うが，チップレベルデザインに対しては抽出時間もさることながら，得られたモデルが膨大な数の回路素子を含むため，シミュレーションに要する時間および必要メモリ量が大きくなるという問題がある [1]．本研究では，寄生 RC 抽出により得られる SPF(Standard Parasitic Format) ファイルを縮小化し，高速かつ少ないメモリ量で，より正確なシミュレーションを行う手法について議論する．

必要な部分の寄生 RC のみ抽出するという機能に関しては，各社の商用 CAD ツールで Selective RC-extraction としてある程度提供されている [2], [3], [4]．しかし，基本的なものとしては，ユーザが抽出したい（あるいは抽出したくない）セル名や信号線名のリストを与える必要があり，大きなデザインでは容易ではない．また，例えば Synopsys の CustomeSim と StarRC は，ある程度自動化された方法として，（寄生 RC 抽出を行う前の）ソースネットリストでまずシミュレーションを行い，指定された以上の電圧変化を起こさない信号線を寄生 RC 抽出の対象から外す，という機能を提供している [5]．しかし，電圧が変化しても当面は着目する必要のない回路ブロックについても寄生 RC 抽出されてしまうとか，逆にバイアス生成回路のように一

[1] 国立情報学研究所・総合研究大学院大学
NII・SOKENDAI, 2-1-2, Hitotsubashi, Chiyoda, Tokyo 101–8430, Japan
[a)] yoneda@nii.ac.jp

ⓒ 2019 Information Processing Society of Japan

定電圧を出力する回路でも寄生 RC 抽出が必要な場合もあり，細かな抽出範囲の調整は容易ではない．

本研究では，商用寄生 RC 抽出ツールにより得られた SPF ファイルを，ユーザが指定した「観測したい（外部）信号線集合」に基づき，不要な寄生 RC 素子を削除し，SPF ファイルを縮小化するというツールを考える．この際，ユーザは観測したい信号線の末端を指定するのみであり，その信号線のシミュレーションに必要な内部信号線や素子はツールにより自動的に選択される．これは，基本的には，指定された信号線から外部入力に到達するまで回路を辿ることで実現する．ただし，シミュレーション結果ができるだけ正確になるように，着目部分から分岐している先の一部も選択する等の工夫を行っている．28nm テクノロジによる 2mm×3mm サイズのチップの寄生 RC 抽出を行い，その SPICE シミュレーションを行ったところ，約 150 時間以上（推定）の計算時間と，約 70GB メモリを必要とするが，提案手法の初期実装ツールを用いて，ある特定の信号線に関する SPF ファイル縮小化を行ったところ，そのシミュレーションに必要な時間，メモリ量を 5 時間弱，10GB 程度に減少させることができた．

## 2. 準備

### 2.1 ツールへの入力

本研究で開発するツールは，(1) ソースネットファイル，(2) SFP ファイル，(3) 観測したい信号先集合，(4) 電源線集合等，を入力とする．ソースネットファイルはレイアウトデザインの元となるもので，LVS(Layout Versus Schematic) をパスしているもの，SPF ファイルはそのソースネットファイルとレイアウトから寄生 RC 抽出により得られたものとする．観測したい信号線は外部出力以外に内部信号線を指定してもよいが，2.3 で述べる信号線の命名規則に従うものとする．電源線集合等については 3.3 で述べる．

### 2.2 ソースネットファイル

ソースネットファイルは，階層的なサブサーキット定義の集合であり，指定されたトップレベルサブサーキット（これを *top* とする）の 1 インスタンスがレイアウトの対象となっている．以後，ソースネットファイル中で定義されているインスタンスおよび信号線をソースインスタンス，ソースネットと呼ぶ．また，ソースインスタンスのうち，サブサーキットとして定義されているもの以外を末端ソースインスタンスと呼ぶ．

### 2.3 SPF ファイル

本ツールで対象とする SPF ファイルは，Synpsys 社の StarRC に，XREF オプションとして XREF=YES 等を与えて生成したものを対象とする．XREF オプションにより，素子や信号線の参照方法が変わるが，ここではこの XREF オプションに基づく命名規則を用いる．

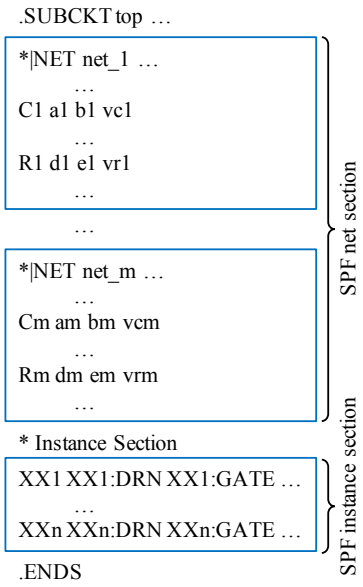


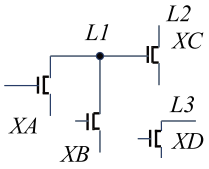

**図 1** SPF TOP の構成

SPF ファイルは，サブサーキット名 *top* を持つ 1 つのフラットなサブサーキットの定義からなる．これを SPF TOP と呼ぶこととする．SPF TOP は，基本的に図 1 に示すように，寄生 RC が定義された部分とインスタンスが定義された部分からなる．本稿では，前者を SPF ネット部，後者を SPF インスタンス部と呼ぶ．

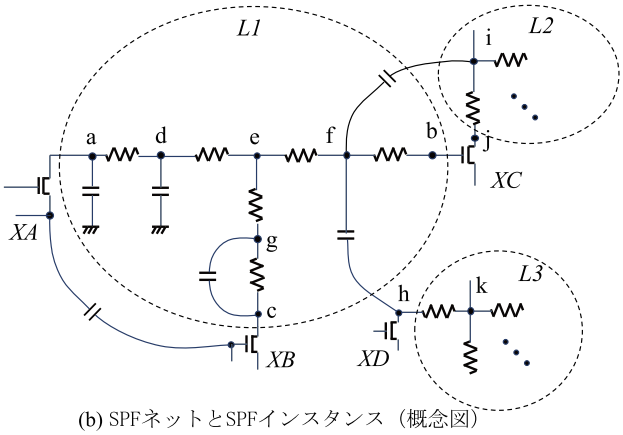
ソースネットファイルにおいて，サブサーキット *top* の中でサブサーキット $S_1$ が $X_1$ としてインスタンス化され，サブサーキット $S_1$ の中でサブサーキット $S_2$ が $X_2$ としてインスタンス化され，$\cdots$，サブサーキット $S_{n-1}$ の中で末端ソースインスタンス $X_n$ が定義されているとする．このとき，この末端ソースインスタンス $X_n$ は SPF インスタンス部では $X_1/X_2/\cdots/X_n$ と参照される．また，この末端ソースインスタンス $X_n$ 中で定義されているソースネット $L$ は，SPF ネット部では $X_1/X_2/\cdots/X_n/L$ と参照される．なお，サブサーキット *top* で定義された末端ソースインスタンスやソースネット $X$ は，SPF ファイルにおいても $X$ として参照される．

SPF ネット部では，“* |NET” から始まるコメント行に信号線名が示されており，その信号線に関連する寄生 RC が定義される．この信号線は基本的にソースネットに対応する（上述のように $X_1/X_2/\cdots/X_n/L$ と参照される）が，ソースネットに対応せず，寄生 RC を付加するために追加された信号線も含まれる．それらを合わせてここでは SPF ネットと呼ぶ．本稿では，SPF ネットを点の集合と考える．例えば図 2 の例では，ソースネット $L1$ に対応する SPF ネットは {a, b, c, d, e, f, g} と考える．また，インスタンスに接続している SPF ネットの点を「端点」，それ以外の SPF ネットの点を「内部点」と呼ぶこととする．図 2

ⓒ 2019 Information Processing Society of Japan




(a) ソースネットとソースインスタンス




(b) SPFネットとSPFインスタンス（概念図）

**図 2** ソースネット・インスタンスと SPF ネット・インスタンス

の例では，SPF ネット $L1$ の端点は，a, b, c, 内部点は d, e, f, g である．

SPF ネットに定義される寄生 R は，その SPF ネットの点間に存在する抵抗となる．図 2 の例では，ソースネット $L1$ に対応する SPF ネットの定義部において，a-d 点間，d-e 点間，$\cdots$ を結ぶ 6 個の寄生 R が定義される．これらの寄生 R によりソースネット $L1$ の接続関係を維持している．一方，SPF ネットに定義される寄生 C は，基本的にはその SPF ネットの点と電源，グランド，自他の SPF ネット，インスタンス等の間に定義される[*1]．異なる SPF ネット間に存在する寄生 C は，いずれかの SPF ネットに定義されることになる．

SPF インスタンス部には，基本的に末端ソースインスタンスに対応するもの（上述のように $X_1/X_2/\cdots/X_n$ と参照される）が定義されるが，SPF ネットと同様に寄生 RC を付加するために追加されたインスタンスも含まれる．これらを合わせて SPF インスタンスと呼ぶ．

アルゴリズムの記述のために以下を定義しておく．

- SPF_net_terminals($L$): SPF ネット $L$ の端点が接続している SPF インスタンスの集合．図 2(b) では SPF_net_terminals($L1$)=$\{XA, XB, XC\}$ となる．
- SPF_net_to_net_by_C($L$): SPF ネット $L$ から寄生 C により接続している SPF ネットの集合．図 2(b) の点 a, c, h のように，寄生 C が SPF ネットの端点に接続している場合は，その接続先は SPF インスタンスと考える．そのため，この例では SPF_net_to_net_by_C($L1$)=$\{L2, 0\}$ となる．なお，0 はグランドを表す．
- SPF_net_to_inst_by_C($L$): SPF ネット $L$ から寄生 C により接続している SPF インスタンスの集合．図 2(b) では，SPF_net_to_net_by_C($L1$)=$\{XB, XD\}$ となる．
- SPF_inst_to_net_by_C($X$): SPF インスタンス $X$ から寄生 C により接続している SPF ネットの集合．図 2(b) では SPF_inst_to_net_by_C($XD$)=$\{L1, \cdots\}$ となる．
- SPF_inst_to_inst_by_C($X$): SPF インスタンス $X$ から寄生 C により接続している SPF インスタンスの集合．図 2(b) では SPF_inst_to_inst_by_C($XA$)=$\{XB, \cdots\}$ となる．

なお，基本的に異なる SPF ネットが寄生 R により接続されることはなく（もし接続されるならそれらは一つの SPF ネットとなる），また異なる SPF インスタンスが寄生 R により直接接続されることもない（もし接続されるならそこに SPF ネットが存在することになる）．同様に，寄生 R により SPF ネットと SPF インスタンスが接続されることもない．

## 3. 縮小化アルゴリズム

次の 4 つのステップにより SPF ファイルを縮小化する．なお，Step 1, 2 はソースネットファイルを対象とし，Step 3, 4 は SPF ネットファイルを対象とする．

### 3.1 Step 1

ソースインスタンスのポートの属性として，{*in, out, inout, power*} を考える．属性 *inout* は，入力も出力も行うポートという意味のほか，入力ポートか出力ポートかが一意に決められない場合に，必要な素子は抽出し損なわないという安全側の処理を行うためにも用いる．初期化として，末端ソースインスタンスが MOSFET の場合，その gate を *in*，bulk を *power* とし，source や drain は入力にも出力にも使うので *inout* とする．また，抵抗やコンデンサ，ダイオードの場合も *inout* で初期化しておく．ユーザが定義したサブサーキットに対してはそのポートに該当する属性を設定する（商用 CAD の Schematic editor を使って設定してもいいし，テキストファイルに記述してツールで読み込むことも可能）．これらの属性をインスタンスの親から子に伝搬させる．この際，属性が *inout* のポートにそれ以外の属性 $a \in$ {*in, out, power*} が伝搬してきた場合は，そのポートの属性を伝搬してきた属性 $a$ に書き換える．また，属性 *power* が伝搬してきた場合は，元のポート属性に関わらず，そのポート属性を *power* に書き換える．それ以外で属性に不一致があった場合は，その旨表示してユーザに修正を求める．

このポート属性に基づいて，次のステップで必要な部分の抽出を行うため，この属性割り当てをできるだけ正確に行うことで，SPF ファイルにおける不必要な部分を精度よく見つけることができ，縮小化を効率よく行うことができ

[*1] 場合によっては該当する SPF ネットとは無関係の点間に定義されたものが含まれることもある．

ⓒ 2019 Information Processing Society of Japan

```
 1: procedure OBTAIN_REACHED
 2:     R_inst ← ∅
 3:     R_net ← 観測したい信号線集合
 4:     N ← R_net
 5:     while N ≠ ∅ do
 6:         C ← ∅
 7:         for net ∈ N do
 8:             for inst ∈ Leaf_insts_of_out_net(net) do
 9:                 R_inst ← R_inst ∪ {inst}
10:                 C ← C ∪ Source_in_nets_of_inst(inst)
11:             end for
12:         end for
13:         N ← C \ R_net
14:         R_net ← R_net ∪ N
15:     end while
16: end procedure
```

**図 3** $R_{inst}$ と $R_{net}$ を求める手続き

る．特に，誤って電源線を（必要な信号が伝搬しているとみなして）辿ってしまうと，多くの不要な SPF インスタンスが抽出されてしまうので，*power* 属性を正確に割り当てることは重要である．

### 3.2 Step 2

ソースネットリスト上で，観測したい信号線集合から入力側に信号線を辿り，到達したソースインスタンス集合 $R_{inst}$ と到達したソースネット集合 $R_{net}$ を図 3 のように求める．ただし，ソースネット *net*，ソースインスタンス *inst* に対し，

- Leaf_insts_of_out_net(*net*): *net* を属性 *out* または *inout* のポートに接続している末端ソースインスタンス集合
- Source_in_nets_of_inst(*inst*): *inst* のポートのうち属性 *in* または *inout* を持つポートに接続されているソースネット集合

とする．このように，基本的に *out* 属性から *in* 属性にソースネットを辿るが，不明なものは含めておくという方針のため *inout* 属性から *inout* 属性へも辿る．一方，上記の理由で *power* 属性のポートには辿らない．

### 3.3 Step 3

前節で得られたソースインスタンス集合 $R_{inst}$ とソースネット集合 $R_{net}$ に対応する SPF インスタンス集合と SPF ネット集合を用いることで，観測したい信号線集合の動作をシミュレーションすることが可能であると考えられる．しかし，例えば図 2 において，$L1 \in R_{net}$, $L2 \notin R_{net}$, $XD \notin R_{inst}$ のとき，同図 (b) の i-f 点間の寄生 C や f-h 点間の寄生 C が無視されてしまうと，それがシミュレーションに与える影響は少なくないと思われる．そこで，シミュレーション精度を高めるために，$R_{inst}$ や $R_{net}$ に隣接するソースインスタンスやソースネットを加えて，それらを拡大する．この手続きを図 4 に示す．行 3 から行 7 は

```
 1: procedure EXPAND_REACHED
 2:     N, I ← ∅
 3:     for net ∈ R_net do
 4:         I ← I ∪ SPF_net_terminals(net)
 5:         I ← I ∪ SPF_net_to_inst_by_C(net)
 6:         N ← N ∪ SPF_net_to_net_by_C(net)
 7:     end for
 8:     for inst ∈ R_inst do
 9:         for net ∈ 全 SPF ネット集合 do
10:             if inst ∈ SPF_net_terminals(net) then
11:                 N ← N ∪ {net}
12:             end if
13:         end for
14:         I ← I ∪ SPF_inst_to_inst_by_C(inst)
15:         N ← N ∪ SPF_inst_to_net_by_C(inst)
16:     end for
17:     Expanded_R_net ← R_net ∪ N
18:     Expanded_R_inst ← R_inst ∪ I
19: end procedure
```

**図 4** $R_{inst}$ と $R_{net}$ を拡大する手続き

$R_{net}$ 中のソースネットに隣接するソースネットやソースインスタンスを加える処理，行 8 から行 16 は $R_{inst}$ 中のソースインスタンスに隣接するソースネットやソースインスタンスを加える処理である．例えば，前述の例で $L2$ は行 6 で追加され，$XD$ は行 5 で追加される．また，例えば $L1 \notin R_{net}$, $XB \in R_{inst}$ のとき，$L1$ は行 11 で追加される．

なお，SPF_net_terminals(*net*) 等 2.3 節の最後で定義した集合は SPF ネットや SPF インスタンスを引数に取るが，図 4 では，ソースネットやソースインスタンスを与えている．厳密には，ソースネットから SPF ネット，あるいは，ソースインスタンスから SPF インスタンスへの変換が必要であるが，簡単化のためここでは省略している，

ところで，前述したように電源線は属性 *power* を持たせることで Step 2 の探索から意図的に外してきた．一方，観測したい信号線に必要なインスタンスにとって，電源線の寄生 R は，例えば MOSFET の source 点の電圧降下によりその出力波形に大きな影響を及ぼす．また，電源線との寄生 C は動作の遅延時間に影響を及ぼす．そこで，上記のようにして得られた $Expanded\_R_{net}$ に，電源線集合を加える．

### 3.4 Step 4

最後に，Step 3 で得られた $Expanded\_R_{inst}$ および $Expanded\_R_{net}$ を用いて，以下の方針により SPF ネット部から不要な寄生 RC を削除し，SPF インスタンス部から不要な SPF インスタンスを削除する．

- 寄生 RC が SPF ネット $a$ の点と SPF ネット $b$ の点に接続されているとき（$a$ と $b$ が同じ場合を含む），$a$ および $b$ に対応するソースネット $a', b'$ が存在し，かつ，$a' \in Expanded\_R_{net}$ および $b' \in Expanded\_R_{net}$ の

ⓒ 2019 Information Processing Society of Japan

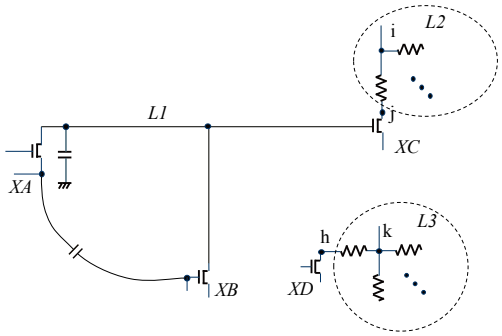



**図 5** SPF ネットの理想信号線への置き換え

ときのみ，その寄生 RC を残す．該当しない寄生 RC は削除する．

- 電源線（グランドの端子'0' も含む）の間に接続されている寄生 C は削除する．
- SPF インスタンス $X$ に対し，対応するソースインスタンス $X'$ が存在し，かつ，$X' \in Expanded\_R_{inst}$ のときのみ，$X$ を残す．該当しない場合は $X$ を削除する．ただし，$X$ を残す場合で，$X$ のポートが SPF ネット $a$ に接続しており，対応するソースネット $a'$ が $a' \notin Expanded\_R_{net}$ のとき，SPF ネット $a$ の各点を抵抗 0 の信号線で結んだ理想信号線を考え，$X$ の対応するポートをその信号線に接続する．これにより，残された SPF インスタンスの接続関係を保持する．例えば，図 2 において，$L1 \notin Expanded\_R_{net}$，$L2, L3 \in Expanded\_R_{net}$，$XA, XB, XC, XD \in Expanded\_R_{inst}$ のとき，図 5 のように $XA, XB, XC$ の対応するポートは，寄生 RC を持たない SPF ネット $L1$ により接続される．

## 4. 評価実験

本節では，提案手法を初期実装して構築したツールに，28nm テクノロジを用いて設計されたミックスドシグナル回路を適用し，効果を評価した結果について述べる．なお，実験環境は，Linux サーバ（CentOS7, Core i7-7820X, 3.6GHz, 128MB Memory），HSPICE G-2012.06-SP2(64bit) であり，ツール自体は Python 2.7 により実装した．

### 4.1 小さな例

まず最初に，当該回路の一ブロックおよびある出力線に着目し，提案手法を適用した．比較対象としては，寄生 RC 抽出したオリジナル SPF ファイルを用いたもの (Original)，Step 3 の手続き EXPAND_REACHED を適用しないもの（ただし，電源線集合の付加は行なっている）(NoExpand)，および手続き EXPAND_REACHED を適用したもの (Expand) とした．得られた SPF ファイルの情報を表 1 に示す．この例では，回路ブロック内の大部分が観測したい信号線に影響を及ぼすため，縮小化は限定的である．手続き EXPAND_REACHED により，約 1 割程度の寄生 RC や SPF インスタンスが追加されている．

**表 1** SPF ファイルの情報 (1)

| | Para. RC # | | SPF inst. # |
|---|---|---|---|
| | R | C | |
| Original | 166,692 | 86,383 | 1,141 |
| NoExpand | 146,951 | 62,947 | 884 |
| Expand | 159,303 | 70,626 | 994 |

シミュレーション波形の一部を図 6 に示す．また，30ns までの過渡解析を行なった場合の HSPICE 実行時間，必要メモリ量，ツールの実行時間等を表 2 に示す．手続き EXPAND_REACHED を実行しない場合 (NoExpand)，本来の波形に比べ，最初の信号の立ち下がり時刻に大きな違いが生じている．また，最後の立ち下がりパルス幅もあまり正確ではない．一方，Expand では，概ね Original に近い波形が得られている．ただし，HSPICE シミュレーション時間や必要メモリ量の削減度合いは小さくなる．

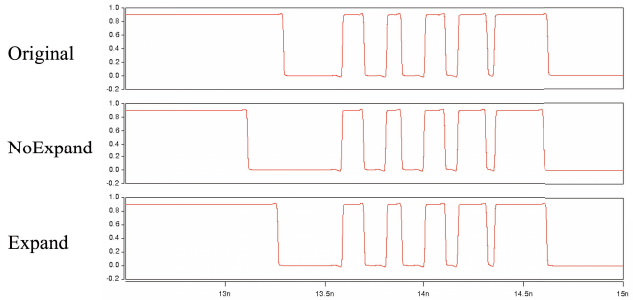



**図 6** 出力波形の比較 (1)

**表 2** シミュレーション等の情報 (1)

| | HSPICE | | Tool | Speed up |
|---|---|---|---|---|
| | Time (s) | Mem. (GB) | Time (s) | |
| Original | 86.7 | 0.89 | - | - |
| NoExpand | 75.2 | 0.80 | 1.2 | 1.13 x |
| Expand | 81.8 | 0.85 | 1.3 | 1.04 x |

### 4.2 チップ全体の例

次に，同様の実験をチップ全体と特定の出力信号線に対して行なった．SPF ファイルの情報を表 3 に示す (Expand2 については後述する)．この例では，観測したい信号線に影響を及ぼす回路ブロックが全体に対して小さいため，大きな縮小化が行えている．

シミュレーション波形の一部を図 7 に示す．現在のところ，Original のシミュレーションは完了していないため，本来の波形との比較はできないが，NoExpand では信号の立ち下がりタイミングが不正確になっているものと思われる．また，30ns までの過渡解析を行なった場合の HSPICE 実行時間，必要メモリ量，ツールの実行時間等を表 4 に

ⓒ 2019 Information Processing Society of Japan

**表 3** SPF ファイルの情報 (2)

| | Para. RC # | | SPF inst. # |
|---|---|---|---|
| | R | C | |
| Original | 37,186,558 | 9,367,735 | 55,634 |
| NoExpand | 6,541,333 | 605,885 | 5,655 |
| Expand | 6,654,977 | 651,742 | 6,278 |
| Expand2 | 6,654,977 | 651,742 | 7,484 |

示す[*2]．Original は推定値である．SPF ファイル縮小化のシミュレーションに与えるインパクトは大きく，30 倍以上のシミュレーション時間の短縮が行えている．また，メモリ量も約 1/7 に抑えられており，メモリ搭載量の小さな計算機でもある程度のシミュレーションが可能となる．NoExpand のほうがシミュレーション時間が長くなっているが，初期 DC 計算結果の違いからくるものと推測できるものの，詳細は不明である．

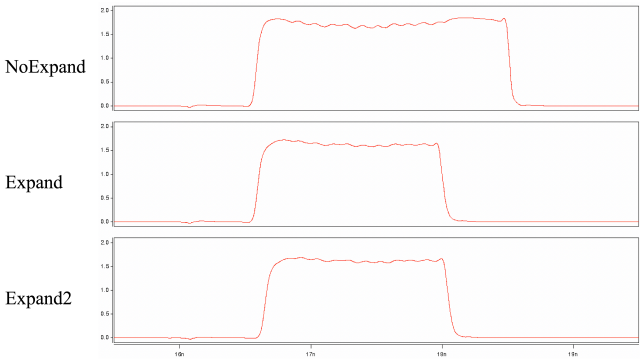



**図 7** 出力波形の比較 (2)

**表 4** シミュレーション等の情報 (2)

| | HSPICE | | Tool | Speed up |
|---|---|---|---|---|
| | Time (s) | Mem. (GB) | Time (s) | |
| Original | > 540,000 | > 70 | - | - |
| NoExpand | 16,573 | 10.7 | 95.1 | > 32.4 x |
| Expand | 15,791 | 10.8 | 103.3 | > 34.0 x |
| Expand2 | 15,275 | 11.0 | 104.0 | > 35.1 x |

なお，NoExpand および Expand バージョンのシミュレーション結果において，アナログ系のコア電源線を調べたところ，Expand バージョンのコア電源線の IR ドロップが NoExpand バージョンの IR ドロップに比べて若干大きいことがわかった．NoExpand バージョンでは Expand バージョンより多くの SPF インスタンスが削除されているため，IR ドロップが小さくなったとも考えられる．この電源電圧の違いによる影響を調べるため，着目する信号線集合が属する回路ブロックと共通のコア電源を使用している SPF インスタンスについては削除しない，というオプションをツールに実装し，このオプションを使用して得られたものを Expand2 とした．追加された SPF インスタンスは，抵抗 0 の信号線で接続関係を維持したのみであるため，表 3 に示すように，Expand2 では SPF インスタンス数のみが増加し，寄生 RC 数は変化していない．図 7 からわかるように，Expand と Expand2 では若干立ち上がりタイミングに差が出ている．回路によっては IR ドロップの影響があり，可能なら SPF インスタンスは削除しないほうが正確なシミュレーション結果が得られそうである．しかし，共通のコア電源を使用している SPF インスタンス数が大きくなる場合は，シミュレーション時間，使用メモリ量の増加をもたらす．

[*2] Orininal の SPF ファイルは非常に大きく（約 3GB），ファイルからの読み込みには時間がかかるため，本ツールでは Redis を用いてメモリ上に格納している．毎回ファイルから読み込む場合は，表 4 のツール実行時間に数分が加算されることになる．

## 5. おわりに

本稿では，観測したい信号線に基づいて SPF ファイルを自動的に縮小化する方法について考察した．観測したい信号線に直接影響を及ぼす信号線集合やインスタンス集合に対し，それらに隣接する信号線集合やインスタンス集合を加えて拡大することで，若干のシミュレーション時間の増大で，より正確なシミュレーションが行えることも示した．提案した手法を初期実装したツールで実験を行ったところ，チップレベルの SPF ファイルに対しては，HSPICE シミュレーション時間で 30 倍以上の高速化が可能であった．

3.2 の Step 2 において，外部入力まで辿る代わりに，指定した信号線集合で探索を停止するようにすることで，回路の内部を選択的に抽出することが可能であり，ユーザの意図した部分のシミュレーションをより効率よく行うことも可能であると考えている．多くの例に適用して本手法の有効性を検証したい．

**謝辞** 本研究の一部は JSPS 科研費 JP15H02254 の助成を受けて実施しました．また，本研究は東京大学大規模集積システム設計教育センターを通し，シノプシス株式会社，日本ケイデンス株式会社の協力により行いました．さらに，本研究を進めるにあたり有益な助言を頂いた，羽生貴弘先生，今井雅先生，吉瀬謙二先生，齋藤寛先生に感謝します．

### 参考文献

[1] W. H. Kao, Chi-Yuan Lo, M. Basel and R. Singh, “Parasitic extraction: current state of the art and future trends,” in Proceedings of the IEEE, vol. 89, no. 5, pp. 729-739, May 2001.
[2] Yousry Elmaghraby, “Efficient parasitic extraction techniques for full-chip verification,”, May 20, 2016. (in `https://www.edn.com`)
[3] “Selective RC-extraction Methods in Guardian LPE for Post-layout Circuit Simulations,” Application Note 2-005, SILVACO.
[4] “Extraction Techniques for High-performance, High-capacity Simulation,” White Paper, Synopsys, Sept. 2009.
[5] CustomSim™ User Guide, Synopsys.

ⓒ 2019 Information Processing Society of Japan