社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2007－ARC－171 (3)
2007－EMB－ 3 (3)
2007／1／22

高位合成システムCCAPにおけるハードウェア関数からのソフトウェア関数の呼び出し

西村啓成[†] 石浦菜岐佐[†] 石守祥之[†]
神原弘之[††] 冨山宏之[†††]

我々は，C プログラムを入力として，指定した関数をソフトウェアから呼び出し可能なハードウェアに合成する高位合成システム CCAP (C Compatible Architecture Prototyper) を開発している．CCAP が合成するシステムでは，ハードウェアに合成した関数 (以下，ハードウェア関数と呼ぶ) を，特別なインターフェースなしにソフトウェア及びハードウェア関数から呼び出すことができる．本稿では，CCAP においてハードウェア関数からソフトウェア関数を呼び出す手法を提案する．また，マルチスレッドあるいはシングルスレッドを用いて実装する例についても示す．このうちシングルスレッドを用いる手法を実装し，シミュレータ上で実際にハードウェア関数からソフトウェア関数を呼び出せることを確認した．

Calling Software Functions from Hardware Functions in High Level Synthesizer CCAP

Masanari Nishimura,[†] Nagisa Ishiura,[†] Yoshiyuki Ishimori,[†]
Hiroyuki Kanbara[††] and Hiroyuki Tomiyama[†††]

We are developing a high-level synthesizer named CCAP (C Compatible Architecture Prototyper), which synthesizes functions in C programs into hardware callable from software. The hardware functions (the functions synthesized into hardware) can be called from software and the other hardware funcstions without designing specific interface in the synthesized system. In this paper, we propose a novel framework in which the hardware functions can call software functions in CCAP, for which we show both multi-thread and single-thread implementation schemes. We verified the correctness of the proposed method (single-thread version) through register transfer level simulation.

1. はじめに

近年，VLSI の回路規模や複雑度は増大の一途をたどっており，その設計効率を向上させることが重要な課題となっている．高位合成技術は，設計者がハードウェアの動作を記述し，そこから自動的にレジスタ転送レベルの回路記述を得る技術であり，設計効率の問題に対する一つの解決策として注目を集めている．入力となる動作記述には，ハードウェア記述言語 (VHDL, Verilog HDL 等) やシステム記述言語 (SpecC, SystemC 等) を用いるのが一般的だが，生産性やシミュレーション速度等の観点から C/C++ のようなプログラミング言語に対する期待が高まりつつある．しかし，C/C++ を入力にとることのできる既存の高位合成ツールは，関数呼び出しやポインタの合成に関して制約が多いという問題があった．

そこで我々は，入力として与えられた C プログラム中の指定された関数を，残りのプログラムから呼び出し可能なハードウェアに合成する高位合成システム CCAP (C Compatible Architecture Prototyper) を開発している．CCAP では，CPU で実行される部分とハードウェアに合成した関数がメモリ空間を共有しており，主記憶を通してデータや制御をやり取りできる．このため，特別なインターフェースを設計することなく関数呼び出しやポインタを合成可能である．

現在 CCAP では，CPU で実行するソフトウェアからのハードウェア関数の呼び出し，およびハードウェア関数からの別のハードウェア関数の呼び出しが可能

† 関西学院大学
Kwansei Gakuin University
†† 京都高度技術研究所
ASTEM
††† 名古屋大学
Nagoya University

である．本稿では，これに加えてソフトウェア関数からハードウェア関数を呼び出す手法を提案する[2]．これにより，例えば内部で動的記憶割り当てを行う関数もハードウェア化が可能となり，高位合成を適用できるプログラムの範囲を大幅に拡大することができる．本稿では，ソフトウェア関数からハードウェア関数を呼び出す具体的な方法として，マルチスレッドを用いる手法とシングルスレッドによる手法を示す．シングルスレッドによる手法を実装し，シミュレーションを行ったところ，実際にハードウェア関数からソフトウェア関数を呼び出せることが確認できた．

2. 関連研究

C/C++ 等のプログラミング言語を入力とする高位合成ツールの研究，開発は近年盛んに行われており，Catapult C Synthesis[3] や eXCite[4] 等，市販されているものも多く存在する．

これらのツールを用いて C プログラム中の関数呼び出しを合成する場合，関数をインライン展開する，あるいは関数をハードウェアに合成し，CPU と各ハードウェアの間に専用インターフェースを設計するのが一般的である．しかし，インライン展開を行うと，複数回の関数呼び出しに対して同じハードウェアが生成されるため，回路面積が増大してしまう．また，インターフェースの設計には設計コストが必要である．

高位合成におけるこの問題を解決する手法として，岡田らは「ソフトウェア/ハードウェア協調設計手法」を特許として公開している[5]．これはソフトウェアとハードウェアの間にデータ授受専用のバッファ RAM を配するもので，個々のハードウェアに対してインターフェースを設計する必要はない．しかし，データの参照渡しができず，ハードウェアからハードウェアを呼び出すこともできないため，合成可能なプログラムのクラスが制限される．

SpC[6] は静的な解析とタグ付けにより，C 言語のポインタの合成を可能にした高位合成ツールである．また，専用ハードウェアを導入することにより `malloc/free` の合成も実現している．しかし，グローバル変数へのポインタの合成や，ソフトウェアとハードウェア関数の間のポインタの受け渡しはサポートしていない．このため，ソフトウェアで確保した動的メモリ領域をハードウェア関数に渡すことはできない．

これらに対し我々の CCAP では，CPU で実行されるソフトウェアとハードウェア関数がメモリ空間を共有しており，これらが主記憶上のグローバル変数を通じてデータや制御をやり取りする．このため，従来の高位合成ツールでは必要だったインターフェースを設計せずとも関数呼び出しを合成することができる．また，SpC では不可能だったソフトウェアとハードウェア関数の間におけるポインタのやり取りを実現できるため，ソフトウェアが確保した動的メモリ領域をハードウェアに渡したり，大規模なオブジェクトを参照渡しで効率よく受け渡すことが可能である．

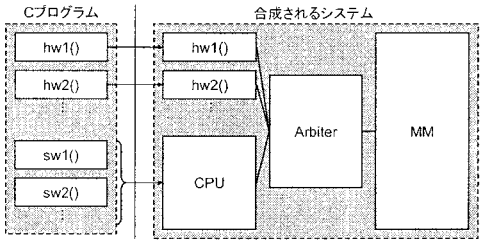


図 1 CCAP が合成するシステム
Fig. 1 A system synthesized by CCAP.

さらに，ハードウェア側からソフトウェアを呼び出す手法に関する研究はこれまで行われて来なかった．本研究で提案する手法により，ハードウェア関数からソフトウェア関数を呼び出すことができるようになれば，ハードウェア関数自身が `malloc` を呼び出して動的メモリ領域を確保することが可能となる．また，ハードウェア関数中に含まれる大規模な処理や例外処理をソフトウェアで実行することにより，回路規模を抑制することもできる．すなわち，高位合成を適用できる C プログラムの範囲を大幅に拡大することができる．

3. 高位合成システム CCAP

3.1 CCAP の合成するシステム

図 1 に，CCAP を用いて合成するシステムの構成を示す．CCAP は，入力となる C プログラム中の一部の関数 (設計者が指定) を，ソフトウェアから呼び出し可能なハードウェア関数に合成する．一方，残りの関数はソフトウェアとしてそのまま CPU で実行する．CPU とハードウェア関数はメモリ空間を共有し，主記憶上に配置するグローバル変数への代入および参照により，データや制御の受け渡しを行う．

CCAP では，CPU や複数のハードウェア関数がそれぞれ主記憶にアクセスするため，アクセス要求を適切に調停する必要がある．これは，各デバイスと主記憶の間にアービタが介在し，アクセス要求を先着順 (同時の場合は CPU を優先) に調停することにより行う．アービタは受理したメモリアクセス要求を主記憶に送り，他のデバイスには stall 信号を送る．

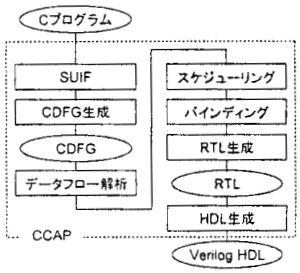
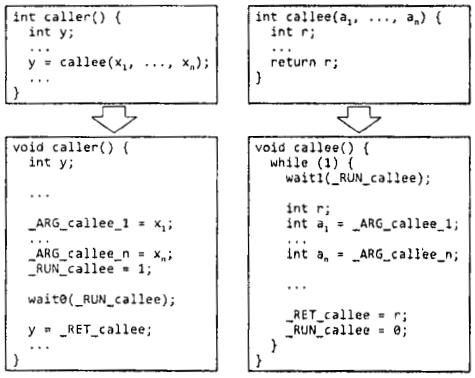


図 2 CCAP の合成の流れ
Fig. 2 Flow of synthesis in CCAP.



(a) 呼び出し側の変換　　(b) 関数側の変換

図 3 ハードウェア関数の呼び出しのための変換
Fig. 3 Transformation for calling hardware function.

CCAP の合成の流れを図 2 に示す. 入力となる C プログラムは SUIF[7] によって中間表現に変換し, そこから CDFG (Control Data Flow Graph) を生成する. これに対してさらにデータフロー解析, スケジューリング, バインディングを行ってレジスタ転送レベルの中間表現を得る. この中間表現を元に Verilog HDL 記述を生成する.

3.2 CCAP におけるハードウェア関数の呼び出し

CCAP におけるハードウェア関数の呼び出しは, 制御やデータの受け渡しのために主記憶上に確保したグローバル変数へのアクセスに変換する. ただし, 再帰呼び出しは扱わないものとする.

図 3 は, 関数 caller からハードウェア関数 callee を呼び出す C プログラムを変換する例である. 呼び出し元である caller はソフトウェアでもハードウェアでもよい. まず, ハードウェア関数 callee に対して, 以下の 3 種類のグローバル変数を作成する.

- _RUN_callee
  callee の実行開始と終了を制御する. 1 が実行中を表し, 0 が非実行中を表す.
- _ARG_callee_1, ..., _ARG_callee_n
  caller から callee に渡す引数を格納する.
- _RET_callee
  callee から caller に渡す返り値を格納する.

図 3 (a) にあるように, caller はまず引数 x_1, ..., x_n の値をグローバル変数 _ARG_callee_1, ..., _ARG_callee_n に代入する. 続いて _RUN_callee に 1 を代入すると, callee が処理を開始する. _RUN_callee の値は callee の処理が終了すると 0 になるので, caller はその値を監視しながら待機する. その後, 返り値をグローバル変数 _RET_callee から受け取り, 残りの処理を続行する.

一方 callee 側では, 図 3 (b) にあるように, まず _RUN_callee を監視しながら, その値が 1 になるまで待機する. その後, _ARG_callee_1, ..., _ARG_callee_n から引数を受け取り, 関数本体の処理を実行する. 処理が終了したら, _RET_callee に返り値を書き込み, _RUN_callee の値を 0 にして, 処理が終了したことを caller に伝える. その後, 再び _RUN_callee の監視に戻る.

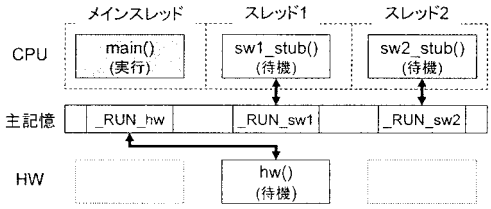
4. ハードウェア関数からのソフトウェア関数の呼び出し

グローバル変数を用いてデータや制御を受け渡せば, 3.2 節と同様の枠組みでハードウェア関数からソフトウェア関数を呼び出すことが可能である. このためには, グローバル変数を監視するプロセスを CPU で実行する必要があるが, これはマルチスレッドを用いて容易に実現できる. また, ソフトウェアからハードウェア関数を呼び出した後の CPU の待機状態を利用すれば, シングルスレッドでも同等の機能を実現可能である.

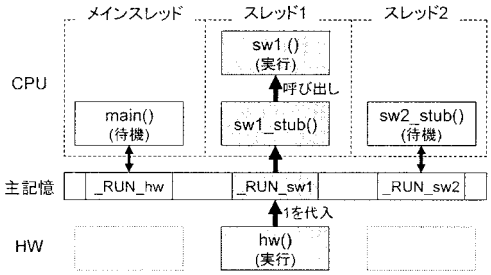
4.1 マルチスレッドによる実現

マルチスレッドが利用可能な環境では, 1 つのソフトウェア関数につき 1 つのスレッドを CPU 上で作成し, ソフトウェア関数の制御用グローバル変数を監視するスタブ関数を各スレッドにおいて実行することにより, ハードウェア関数からソフトウェア関数を呼び出す手法を提案する. スタブ関数は制御用グローバル変数の監視の他, 引数用グローバル変数からの引数の受け取り, 通常の方法でのソフトウェア関数の呼び出し, 返り値用グローバル変数からの返り値の受け取りを行う.

図 4 (a) に, マルチスレッドを用いた制御用グローバ



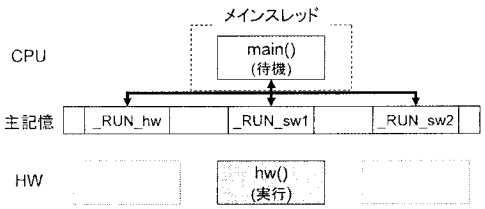
(a) マルチスレッドによるグローバル変数の監視



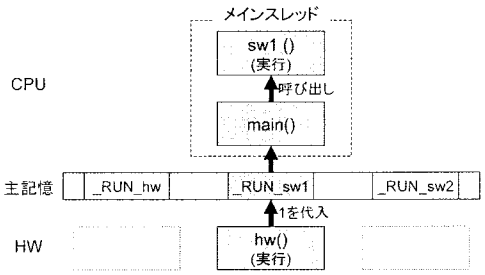
(b) ハードウェア関数からのソフトウェア関数呼び出し

図 4 マルチスレッドによるハードウェア関数からのソフトウェア関数の呼び出し

Fig. 4 Multi-thread scheme of calling software functions from hardware functions.



(a) シングルスレッドによるグローバル変数の監視



(b) ハードウェア関数からのソフトウェア関数呼び出し

図 6 シングルスレッドによるハードウェア関数からのソフトウェア関数の呼び出し

Fig. 6 Single-thread scheme of calling software functions from hardware functions.

```
void callee_stub() {
  while (1) {
    while (!_RUN_callee) {}
    _RET_callee = callee(_ARG_callee_1, ...);
    _RUN_callee = 0;
  }
}
```

図 5 スタブ関数の例

Fig. 5 An implementation example of stub function.

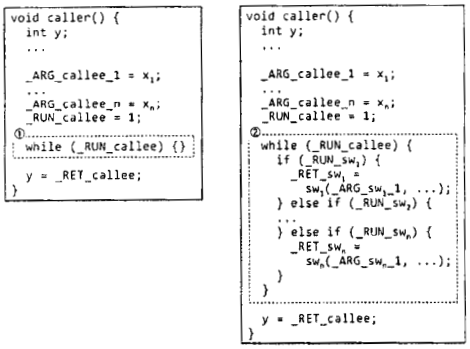
ル変数の監視について示す．この図において，`main()`，`sw1_stub()`，`sw2_stub()` は CPU 上で実行するソフトウェア関数，`hw()` はハードウェアに合成する関数である．CPU 上には，メインスレッドの他に 2 つのスレッドが存在し，これらはそれぞれ `sw1()` のスタブ関数 `sw1_stub()` と，`sw2()` のスタブ関数 `sw2_stub()` を実行している．各スタブ関数はバックグラウンドで動作し，制御用グローバル変数を監視しながら，その値が 1 になるまで待機する．ここで，図 4 (b) にあるように `hw()` が `_RUN_sw1` に 1 を書き込むと，それを監視する `sw1_stub()` がこれを検知し，`sw1()` を呼び出す．

図 5 は，各スレッドで実行するスタブ関数の例である．関数 $\texttt{callee}_{\texttt{stub}}$ は関数 `callee` のスタブ関数であり，`callee` の制御用グローバル変数 `_RUN_callee` を監視する．この値が 1 になると，$\texttt{callee}_{\texttt{stub}}$ はグローバル変数 `_ARG_callee_1`, ..., `_ARG_callee_n` の値を引数として，通常の関数呼び出しと同じように `callee` を呼び出す．返り値はグローバル変数 `_RET_callee` に代入する．その後，`_RUN_callee` に 0 を代入して `caller` に実行終了を通知し，再び `_RUN_callee` の監視に戻る．CPU 上で実行される他のソフトウェア関数からは，スタブ関数を介さずに呼び出すことにより，そのまま呼び出すことができる．

4.2 シングルスレッドによる実現

マルチスレッドによる実現手法は，OS やライブラリ等の利用が前提となる．また，複数のスレッドを常時実行させておくオーバヘッドが性能上の問題となる場合もある．ハードウェア関数の実行終了を待ち合わせるための CPU の待機状態を利用することにより，シングルスレッドでハードウェアからソフトウェア関数を呼び出す手法を提案する．

図 6 に，CPU の待機状態を用いてハードウェア関数からソフトウェア関数を呼び出す手法を示す．ソフトウェア側からハードウェア関数 `hw()` を呼び出すと，ハードウェア関数が実行を開始し，CPU はハードウェア関数の制御用グローバル変数を監視しながら待機状態に入る．この待機状態を，図 6 (a) にあるように，複数のソフトウェア関数の制御用グローバル変数に対する監視を繰り返す処理に書き換える．ハードウェア



(a) 従来の待機状態　　(b) 待機状態を利用した監視

図 7　待機状態を利用したグローバル変数の監視
Fig. 7　Polling of global variables during the wait state.

関数からいずれかのソフトウェア関数に対して, その制御用グローバル変数に 1 が代入されると, CPU 側でこれを検知し, 引数用グローバル変数の値を引数として, 該当するソフトウェア関数を呼び出す. ソフトウェア関数の実行が終了すると, 受け取った返り値をグローバル変数に書き込み, 制御用グローバル変数の値を 0 にして, ソフトウェア関数の実行終了を示す. その後, CPU は再び待機状態に戻り, `hw()` が終了するまで, ソフトウェア関数の制御用グローバル変数の監視を続ける.

図 7 は CPU の待機状態中に実行する監視処理の実装例である. 図 7 (a) は従来の待機状態, (b) は監視処理を加えた新しい待機状態である. ① の `while` 文はハードウェア関数 `callee` の実行終了待ち状態を busy loop で実装したものである. 従来の CCAP の実装では, ハードウェア関数の制御用グローバル変数に対する値のテストを繰り返すのみであった. これを図 7 (b) ② のように, `callee` から呼び出され得るソフトウェア関数の制御用グローバル変数 $\_RUN\_sw_1$, ..., $\_RUN\_sw_n$ をテストする処理を `while` 文の本体に加える. もしいずれかの値が 1 ならば, 引数用グローバル変数の値を引数として, 該当するソフトウェア関数を通常の方法で呼び出す. ② のループはメインスレッドで実行され, 呼び出されるソフトウェア関数の数にかかわらず 1 つである. これにより, マルチスレッドによる実現手法と同等の機能をシングルスレッドで実現することが可能となる.

4.3　提案手法の利点と欠点

マルチスレッドによる実現手法は, OS やライブラリによるサポートが必要であることや, 複数のスレッドをバックグラウンドで常時実行させておく必要があることなどから, CPU 性能に余裕があるシステムに向く. 特にマルチ CPU を備えたシステムの場合, 複数のスレッドの実行を各 CPU に振り分けることで, パフォーマンスの低下を抑えながらハードウェア関数からのソフトウェア関数の呼び出しを達成することが可能である. 一方シングルスレッドによる実現手法は, 特にライブラリ等を必要としないため, 環境を選ばないという利点がある. また, CPU の待機状態を活用するため, CPU で実行される他のソフトウェアプロセスのパフォーマンスを損なうこともない.

```
main() {
  ソート対象となる配列 a の初期化;
  bucket_sort(a) の呼び出し;
}

bucket_sort(a[]) {
  バケット数を求める;
  malloc を呼び出してバケットの領域を確保;
  バケットにソート情報を生成;
  バケット内の情報に従って配列をソート;
}
```

図 8　実験用 C プログラムの概要
Fig. 8　Outline of experimented C program.

5. 実　　験

本稿で提案した手法のうち, シングルスレッドによる実装手法に基づき, ハードウェア関数からソフトウェア関数を呼び出す実験を行った.

図 8 に実験用の C プログラムの概要を示す. `main` 関数を CPU で実行し, `bucket_sort` 関数をハードウェア化して `main` から呼び出した. CPU には MIPS R2000 (ソフトコア) を用い, `bucket_sort` を手設計でハードウェア化したもの及びアービタとともに RT レベルでシミュレーションを行った. `bucket_sort` 内部から呼び出す `malloc` は, MIPS R2000 用の C ライブラリが利用できないため, `malloc` と同様の動作を行うソフトウェア関数を実装した. 以上のシステムを ModelSim (XE II 5.8c) でシミュレーションし, `main` から起動された `bucket_sort` が `malloc` を呼び出してバケット用領域を動的に割り当て, 正しくソートを行うことを確認した.

6. むすび

本稿では, 高位合成システム CCAP においてハードウェア関数からソフトウェア関数を呼び出す方法を提案し, そのマルチスレッドによる実装法とシングルスレッドによる実装法を示した. また, シングルスレッドによる手法を実装し, 実際にハードウェア関数からソフトウェア関数を起動できることをシミュレーショ

ンにより確認した.

今後, ソフトウェア関数からハードウェア関数を呼び出すための処理を自動的に生成できるように, 本稿で述べた手法を CCAP に実装する予定である. また, 現在 CPU の待機状態は busy loop により実現している. これを, 割り込みを用いて実装することも検討していきたい.

謝辞 本研究に際し, シミュレータによる実験に御協力頂いた立命館大学の梅原直人氏, 中谷嵩之氏に感謝致します. また, 御討論頂いた京都大学の杉原有理氏, 及び関西学院大学石浦研究室の諸氏に感謝致します.

参 考 文 献

1) 西口 健一, 石浦 菜岐佐, 西村 啓成, 神原 弘之, 冨山 宏之, 高務 祐哲, 小谷 学: "ソフトウェア互換ハードウェアを合成する高位合成システム CCAP における変数と関数の扱い," 信学技報, VLD2005-12 (Dec.2005).
2) 西村 啓成, 石浦 菜岐佐, 石守 祥之, 神原 弘之, 冨山 宏之: "高位合成システム CCAP におけるハードウェア関数からのソフトウェア関数の呼び出し," 情処関西支部大会, C-05 (Oct.2006).
3) http://www.mentor.com/.
4) http://www.yxi.com/.
5) 岡田 和久: "ソフトウェア/ハードウェア協調設計手法," 公開特許広報, 特開 2003-114914 (2003).
6) L.Séméria, K.Sato, and G.De Micheli: "Synthesis of Hardware Models in C With Pointers and Complex Data Structures," in *IEEE Trans. VLSI Systems*, vol.9, no.6, pp.743–756 (Dec.2001).
7) http://suif.stanford.edu/.