A-02

高位合成系 ACAP を用いた モーターの浮動小数点モデルの FPGA 上での実行

FPGA Simulation of Motor in Floating Point Precision Using Binary Synthesizers ACAP

竹林 陽 † 伊藤 直也 † 田村 真平 ‡
Hinata Takebayashi Naoya Ito Shimpei Tamura
神原 弘之 § 石浦 菜岐佐 †
Hiroyuki Kanbara Nagisa Ishiura

1 はじめに

自動車の電装系には, 動力を発生する DC モーター, DC モーターをソフトウェアに従って制御する CPU, CPU からの指示に従って DC モーターの ON–OFF 動作を行うアナログ回路 (スイッチングトランジスタ) が多数存在する. このような電装系の設計期間を短縮しつつ設計品質の向上を図るため, CPU やモーターの実機が揃わない設計の初期段階から, 正常動作時あるいはモーターの故障状態の動作を再現できる, モデルベースのシミュレーション技術への要請が高まっている [1].

図 1 (a) に示すように, PC 上で「CPU の動作を命令レベルで模擬するシミュレータ (ISS) 」と「モーターの動作を電磁界解析に基づき模擬するシミュレータ」を組合せることにより, モーターの CPU による制御：HIL (Hardware in the loop) をシミュレーションすることができる. しかし, この方法では 2 つのソフトウェアの速度, およびそれらの間の通信等によりシミュレーション速度が律速され, モーターや CPU の様々な動作を短時間に再現できないことが課題になっている.

このうち, CPU については, ハードウェア記述言語を用いたレジスタ転送レベル (RTL) の記述から FPGA 向けに論理合成とマッピングを行うことにより, 組込み用の製品とほぼ同等の速度で動作させるプロトタイピングが可能になってきている. CPU と同様に, 実機と同等の速度でその振舞いを再現できるモーターやアナログ回路のモデリング技術を確立できれば, 図 1 (b) のように全体の実行を FPGA 上で行うことにより, シミュレーション速度を格段に向上させることができる.

本研究では, モーターの伝達関数によるモデルを高位合成系 ACAP を用いて FPGA 上に実現することにより, 高速なシミュレーションを可能にする手法を提案する. 整数演算のみのプログラムを対象としていた ACAP を浮動小数点に対応する拡張を行うことで, モーターのシミュレーションに用いられる浮動小数点演算を実行可能にした. モーターのシミュレーションプログラムを実行するために, gcc の浮動小数点エミュレーションライブラリである soft-float をリンクし, (1) soft-float 関数全体のハードウェア化, (2) soft-float 関数の呼び出し部分の浮動小数点演算器へのマッピングを行う. これにより, FPGA 上で浮動小数点対応していない CPU であっても ACAP が生成したハードウェアと組み合わせることにより, CPU を変更を加えることなく浮動小数点演算を高速に行うことができる.

DC モーターを伝達関数によってモデル化することによりシミュレーションを行うプログラムをハードウェア化し, 実験を行った. 手法 (1) では, CPU の回路規模に対して約 134.0% の回路規模の増大で, 実行サイクル数を 36.5% 削減することができた. 手法 (2) は, 実装を行なっていないので見積もり値ではあるが, CPU の回路規模に対して約 103.1% の回路規模の増大で, 実行サイクル数を 95.4% 削減することができると考えられる.

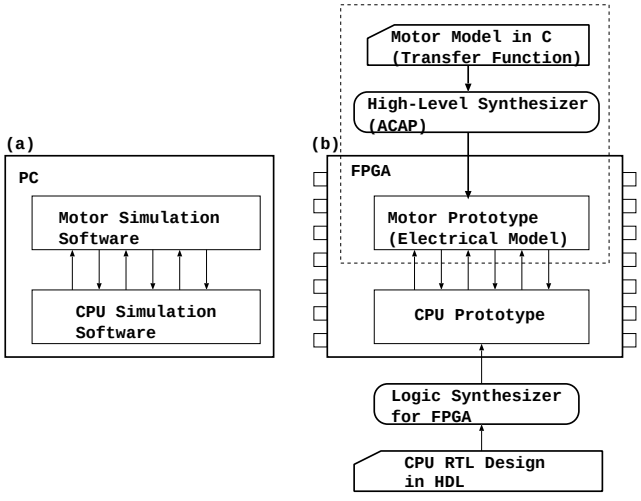

図 1 モーター制御のシミュレーション

† 関西学院大学, Kwansei Gakuin University
‡ 株式会社エムエスティ, Marking System Technology Co., Ltd.
§ 京都高度技術研究所, ASTEM RI

2 モーターのシミュレーション

2.1 伝達関数によるモデル化

モーターやアクチュエータの検証は, 多くの場合伝達関数を用いたシミュレーションにより行うことができる.

図 2 はブラシ付き DC モーターの等価回路を表したものである. 図 3 は図 2 をモーターパラメータによる定数と 2 個の 1 次遅れ要素によるブロック図で表したものである.

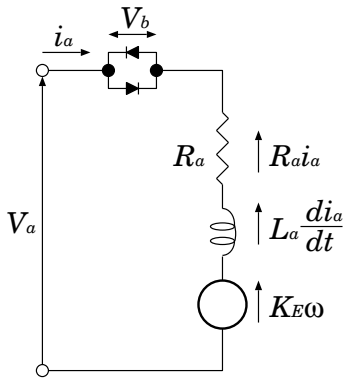

図 2 ブラシ付き DC モーターの等価回路 [2]

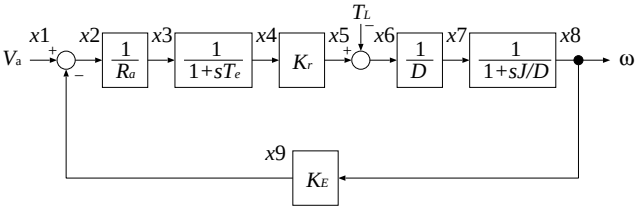

図 3 ブラシ付き DC モーターの伝達関数のブロック図 [2]

ここでは, s: ラプラス演算子, T_e: 電気的時定数 ($T_e = L_a/R_a$), V_b: ブラシの電圧降下, R_a: 巻線抵抗, L_a: 巻線インダクタンス, D: 粘性制動係数, J: 慣性モーメント, K_E: 逆起電力定数, K_T: トルク定数 である. この伝達関数では, 任意の大きさ, 任意のタイミングで負荷トルクを印加でき, そのときの電機子電流や回転速度を調べることができる [2].

2.2 モーターにおける実数値の計算

モーターのシミュレーションモデルは実数値の計算を多く使用しているが, これらは浮動小数点演算によって処理が行われる. CPU 上で浮動小数点演算を行う場合, 専用の演算器を用いる手法と整数演算のみでエミュレーションを行う手法の 2 通りがある.

前者は, FPU (Floating Point Unit) と呼ばれる処理装置を用いて専用命令を実行する. FPU は, アーキテクチャによって, 周辺機器として実行されるものもあれば, コプロセッサとして CPU と密結合して実行されるものもある. MIPS R3000 の場合, R3010 というチップとして実装されており, CPU と密結合して使用される. R3010 は, 加算ユニット, 乗算ユニット, 除算ユニット, 指数ユニット等の各浮動小数点演算を実行するユニットと, レジスタユニットから成る. FPU は, 命令メモリ中の浮動小数点演算をすべて実行し, 結果をレジスタユニットに格納する. CPU は, FPU レジスタ転送命令により, その結果を取得することができる.

後者は, 浮動小数点エミュレーションライブラリを用いて, 浮動小数点演算をすべて整数演算のみで処理する. gcc で soft-float ライブラリを使用する場合, 各演算に対応する関数 (表 1) を呼び出すことにより計算が行われる. 計算のオペランドは, 引数として関数に渡し, 計算結果は関数の返り値として取得する.

表 1 MIPS R3000 の浮動小数点演算に対応する soft-float 関数

関数名	機能
__addsf3	v0 = a0 + a1
__subsf3	v0 = a0 - a1
__mulsf3	v0 = a0 * a1
__divsf3	v0 = a0 / a1
__negsf2	v0 = -a0
__fixsgsi	v0 = (int) a0
__fixunsfsi	v0 = (unsinged int) a0
__floatsisf	v0 = (float) a0
__floatunsisf	v0 = (float) a0
__cmpsf2	v0 = a0 <=> a1
__unordsf2	v0 = a0 a1
__eqsf2	v0 = (a0 == a1) ? 1 : 0
__nesf2	v0 = (a0 != a1) ? 0 : 1
__gesf2	v0 = a0 <=> a1
__ltsf2	v0 = a0 <=> a1
__lesf2	v0 = a0 <=> a1
__gtsf2	v0 = a0 <=> a1

FPU を用いる手法では, 演算を高速に実行することができる. しかし, 演算器を追加することにより回路規模が増大する. さらに専用命令を追加するため, 命令セットを変更する必要がある.

浮動小数点演算をエミュレーションする手法では, 命令セットを変更することなく, 浮動小数点演算を扱うことができる. しかし, 例えば, MIPS R3000 の場合では浮動小数点の加算 (__addsf3) を一回を行うのに約 120 サイクル必要になるため, soft-float の関数を用いて浮動小数点演算を行うと実行時間が非常に長くなる. したがって, 浮動小数点演算を多く使用するプログラムではその分だけ処理に時間がかかる.

3 バイナリ合成

高位合成は, C 等の高級言語からハードウェアの回路記述を合成する技術である. その中でも, アセンブリや機械語を入力として用いるもの [3] はバイナリ合成とも呼ばれる. アセンブリや機械語を入力として用いることにより, ポインタや構造体などのより広範なプログラムを容易にハードウェアに合成することができる.

高位合成系 ACAP [4] は, MIPS R3000 の機械語プログ

ラムを入力として, これを 32bit の整数演算命令を並列に実行可能なハードウェアに変換することにより, 処理の高速化を実現する. ACAP は, 図 4 の流れによりハードウェアの合成を行う. 機械語プログラムを CDFG (Control Data Flow Graph) に変換し, 最適化, スケジューリング, バインディング処理を行い, Verilog HDL を出力する.

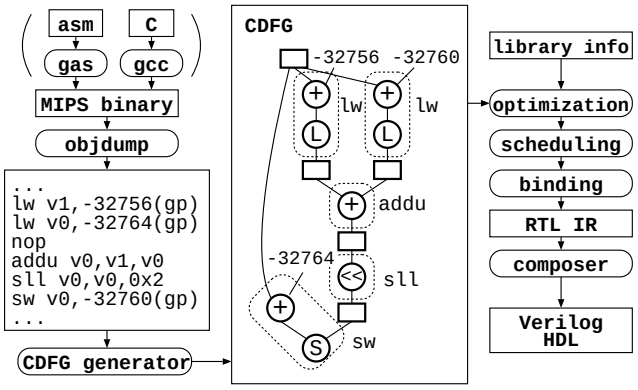

図 4　ACAP での高位合成の処理の流れ [4]

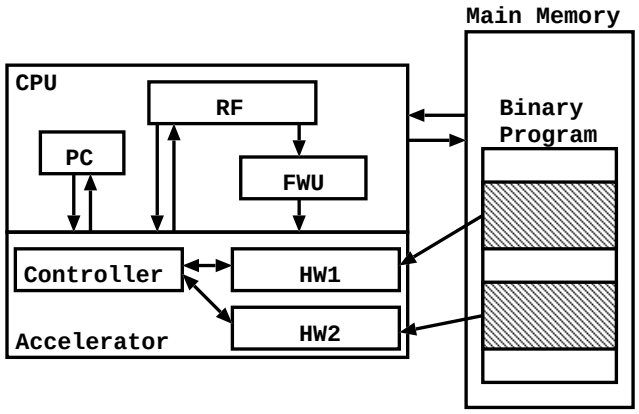

図 5　CPU 密結合型アクセラレータ

高位合成系 ACAP のアクセラレータモード [5] では, リンク済み実行可能コードの一部を指定して, 等価な動作をより高速に行うハードウェアを生成する. アクセラレータは CPU の資源に直接アクセスすることにより制御やデータの受け渡しを高速に行うことができる.

アクセラレータの構成を図 5 に示す. アクセラレータは CPU のプログラムカウンタ (PC) の値を常に監視し, その値がアクセラレータ化した区間の先頭番地に達すると処理を開始する. アクセラレータは処理を開始すると PC の値を固定し, CPU に NOP 命令を供給し続ける. アクセラレータは処理が完了すると, PC に復帰番地を書き込み, 制御を CPU に戻す. アクセラレータはメインメモリや CPU のレジスタファイル (RF), フォワーディングユニット (FWU) に直接アクセスすることにより, CPU とデータ共有を行う.

4　浮動小数点演算のハードウェア化

本研究では, 高位合成系 ACAP を単精度の浮動小数点演算に対応する拡張を行い, モーターの伝達関数によるモデルを FPGA 上に実現し, シミュレーションの高速化を図る. CPU で浮動小数点演算を実行する場合, FPU を用いる場合 (図 6 (a)) と soft-float ライブラリとリンクして整数演算として扱う場合 (図 6 (b)) があるが, このうちの後者を対象にした ACAP の拡張を行う. soft-float とリンクしたプログラムを対象として, はじめに soft-float ライブラリ全体をアクセラレータ化する手法 (図 6 (1)) を示し, 次に soft-float ライブラリの関数を呼び出す命令を検出し, それぞれ専用の演算器にマッピングする手法 (図 6 (2)) を示す.

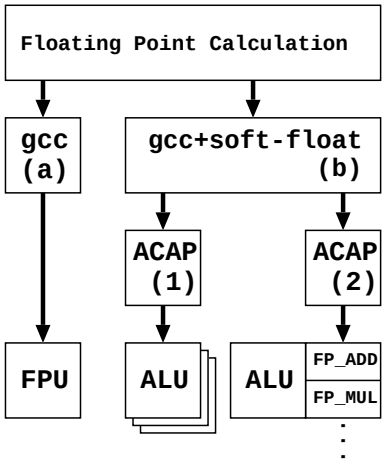

図 6　浮動小数点演算の実行

4.1　soft-float ライブラリのアクセラレータ化

提案手法では, soft-float ライブラリ全体をアクセラレータとして合成する. その手法を図 7 に示す. soft-float ライブラリとリンクした実行可能コードから, soft-float 関数の処理部分をすべて選択する. 選択した部分を高位合成系 ACAP によって, 単一のアクセラレータとして合成し, CPU に接続する. soft-float 関数本体はアクセラレータとして実行され, soft-float 関数の呼び出しを含む残りの部分は CPU により実行される.

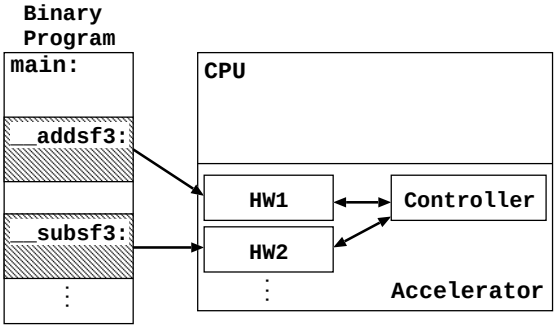

図 7　soft-float 関数全体のアクセラレータ化

4.2 関数呼び出しの浮動小数点演算器へのマッピング

soft-float 関数の呼び出し部分を検出して, 対応する演算器にマッピングする. 図 8 にその変換手法を示す.

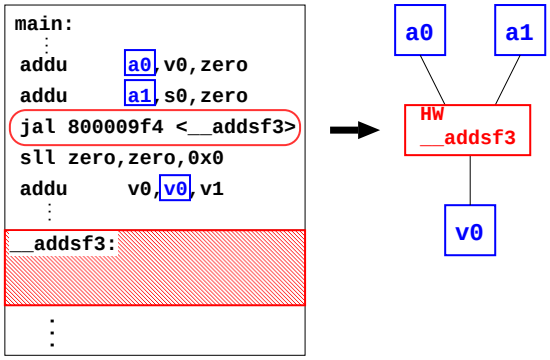

図 8 soft-float 関数呼び出しの浮動小数点演算器へのマッピング

CDFG 生成時に soft-float 関数を呼び出す命令を, 引数のレジスタを入力とし, 返り値のレジスタを出力とする浮動小数点演算に変換する. 引数と返り値のレジスタの判断は, MIPS R3000 のコンパイラ規約から前者を a0, a1, 後者を v0 とする. 変換した浮動小数点演算をバインディング時に対応する浮動小数点演算器に割り当てる.

5 実験と考察

5.1 実験

本手法で拡張を行った高位合成系 ACAP を用いてハードウェアを合成し, 実験を行った. 図 3 のブロック図に基づき, ブラシ付き DC モーターのシミュレーションを行うプログラムを soft-float ライブラリとリンクし, soft-float 関数全体, および DC モーターのシミュレーション部分をアクセラレータに合成した. CPU およびハードウェアは FPGA Xilinx Spartan3E 上に合成された Verilog HDL を論理合成ツール Xilinx ISE 14.3 によって論理合成し, 性能評価を行った. CPU は MIPS R3000 互換プロセッサ [6] を使用した. 浮動小数点演算器のうち, 3 サイクルの加減算器と 1 サイクルの乗算器は独自で作成したものを用い, 8 サイクルの除算器と 1 サイクルの比較演算器は Xilinx Core Generator により作成したものを用いた.

実験に用いたプログラムを図 9 に示す [2]. このプログラムは伝達関数を常微分方程式により記述し, 数値解析によりその振る舞いをシミュレーションする. 数値解析は,4 次のルンゲ・クッタ法を用いて近似解を求めることにより行う. 今回の実験では各パラメータを, 巻線抵抗 R_a=1Ω, 巻線インダクタンス L_a=8mH, 逆起電力定数 K_E=0.28Vs/rad, トルク定数 K_r=0.28Nm/A, 慣性モーメント J=0.005kgm^2, 粘性制動係数 D=0.0002Nms/rad, 計算時間間隔 dt=5ms, ループ回数 te/dt=200 と設定した. アクセラレータの合成では, ALU, 乗除算器の資源制約はそれぞれ 9 個, 2 個に設定し, チェイニングは行なっていない.

結果を表 2 に示す. Target の CPU は soft-float ライブラリとリンクした機械語プログラムを MIPS で実行したものである. CPU+ACC (1) は, soft-float ライブラリ全体をアクセラレータ化したものを CPU と接続したものである. CPU+ACC (2) は, 図 9 の `main` 関数と `rk4` 関数を対象として, その中の soft-float 関数の呼び出し部分を検出して演算器にマッピングし, アクセラレータ化したものを CPU と接続したものである. Cycle, Slice, Delay は, それぞれ実行サイクル数, スライス数, 遅延時間を表す. ただし, CPU+ACC (2) は未実装であるため, (2) のアクセラレータは, プログラム中の soft-float 関数呼び出し部分をすべて NOP 演算に置き換えたものから合成されており, 表 2 で示す結果はすべて見積もり値である. Cycle は, 各 DFG のステップ数から算出した. Slice は, CPU とこのアクセラレータのスライス数に, プログラム中で用いられる各浮動小数点演算器のスライス数を加えることにより算出した. Delay は, 上記の方法により生成したアクセラレータを CPU と接続したものから計測した.

実際のスライス数は, 浮動小数点演算器に対するデータ

```
01: float rk4(float xin, float xold,
02:           float tau, float dt){
03:   float k1, k2, k3, k4, xout;
04:
05:   k1 = dt * (xin - xold) / tau;
06:   k2 = dt * (xin - (xold + k1 /2.0)) / tau;
07:   k3 = dt * (xin - (xold + k2 /2.0)) / tau;
08:   k4 = dt * (xin - (xold + k3)) / tau;
09:
10:   xout = xold+(k1+2.0*k2+2.0*k3+k4)/6.0;
11:   return(xout);
12: }
13:
14: int main(void)
15: {
16:   float x1, x2, x3, x4, x5, x6, x7, x8, x9;
17:   float Ra, La, K, J, D, Va, TL, TLs, TLe;
18:   float ts, te, dt, taue, tauj;
19:   float t, Tload;
20:
21:   x4 = x8 = x9 = 0.0;
22:
23:   Ra = 1.0;          //電機子抵抗
24:   La = 0.008;        //電機子インダクタンス
25:   K = 0.28;          //モータ定数 (KE = kT = K)
26:   J = 0.005;         //不可を含む慣性モーメント
27:   D = 0.0002;        //粘性制動係数
28:
29:   Va = 80.0;         //モータ駆動電圧
30:   TL = 1.4;          //付加トルク
31:   ts = 0.0;          //シミュレーション開始時間
32:   te = 0.1;          //シミュレーション終了時間
33:   TLs = 0.4;         //負荷印加開始時間
34:   TLe = 0.7;         //負荷印加終了時間
35:
36:   dt = 0.0005;       //計算時間間隔
37:   taue = La / Ra;
38:   tauj = J / D;
39:
40:   for(t = dt; t<te; t+=dt){
41:     if(t > TLs && t < TLe) Tload = TL;
42:     else Tload = 0.0;
43:
44:     x1 = Va;
45:     x2 = x1 - x9;
46:     x3 = x2 / Ra;
47:     x4 = rk4(x3, x4, taue, dt); //電機子電流の計算
48:     x5 = x4 * K;              //モータ発生トルクの計算
49:     x6 = x5 - Tload;              //負荷トルクを印加
50:     x7 = x6 /D;
51:     x8 = rk4(x7,x8,tauj,dt);    //回転角速度の計算
52:     x9 = x8 * K;
53:   }
54:   return 0;
55: }
```

図 9 モーターのシミュレーションプログラム [2]

表 2　実験結果

Target	Cycle	Slice	Delay[ns]
CPU	2461938 (100.0%)	3156 (100.0%)	25.113
CPU+ACC (1)	1563927 (63.5%)	7384 (234.0%)	25.906
CPU+ACC (2)	112324 (4.6%)	6411 (203.1%)	25.669

パスが加わるため, 表 2 に示す結果よりも増大すると考えられる. アクセラレータの遅延は, チェイニングを行わない限り, 基本的に CPU の遅延を超えることはないため, 実際の結果もほぼ同様の値になると考えられる.

CPU+ACC (1) は, CPU の回路規模に対して約 134.0% の回路規模の増大で, 実行サイクル数を 36.5% 削減することができた. CPU+ACC (2) は, CPU の回路規模に対して約 103.1% の回路規模の増大で, 実行サイクル数を 95.4% 削減することができると推測される.

5.2 考察

提案手法 (1) は, 浮動小数点演算器のない環境において, soft-float 関数の実行を高速化することができる. 提案手法 (2) は, 必要最低限の浮動小数点演算器のみを用いることにより回路規模の増大を防ぎ, 命令セットを変更することなく, 浮動小数点の計算を高速に行うことができる.

6 むすび

本研究では, モーターの伝達関数によるモデルを高位合成系 ACAP を用いて FPGA 上に実現し, 高速なシミュレーションを可能にした. 今回の実験では, モーターの制御に関する計算を 5ms 間隔で行うように設定しているが, soft-float 関数の呼び出し部分を演算器にマッピングする手法を用いることにより, 1.5ms 間隔まで短縮することができる.

謝辞

本研究に関して有益な御助言を頂いた元立命館大学の中谷嵩之氏, 元京都大学の矢野正治氏に感謝致します.

参考文献

[1] Christian Dufour, Jean Belanger, Simon Abourida, and Vincent Lapointe: “FPGA-Based Real-Time Simulation of Finite-Element Analysis Permanent Magnet Synchronous Machine Drives,” *in Proc. Power Electronics Specialists Conference (PESC 2007)*, pp. 909–915 (June 2007).

[2] 高橋久: C 言語によるモーター制御入門講座 〜SH マイコンで学ぶプログラミングと制御技法〜, 電波新聞社 (Oct. 2007).

[3] Greg Stitt and Frank Vahid: “Binary synthesis,” *ACM Trans. on Design Automation of Electronic Systems*, vol. 12, no. 3, article 34 (Aug. 2007).

[4] Nagisa Ishiura, Hiroyuki Kanbara, and Hiroyuki Tomiyama: “ACAP: Binary Synthesizer Based on MIPS Object Codes,” *in Proc. International Technical Conference on Circuit/Systems Computers and Communications (ITC-CSCC 2014)*, pp. 725–728 (July 2014).

[5] 田村真平, 石浦菜岐佐, 神原弘之, 冨山宏之: “CPU 密結合型アクセラレータの機械語プログラムからの自動合成,” 電子情報通信学会技術研究報告, VLD2013-133 (Jan. 2014).

[6] 神原弘之, 金城良太, 矢野正治, 戸田勇希, 小柳滋: “パイプラインプロセッサを理解するための教材: RUE-CHIP1 プロセッサ,” 情報処理学会関西支部大会, A-09 (Sept. 2009).