A-01

割り込み駆動のモータ制御プログラムからの高位合成

High-Level Synthesis from Interrupt-Driven Motor Control Programs

大迫 裕樹 † 神原 弘之 ‡ 石浦 菜岐佐 †
Yuuki Oosako Hiroyuki Kanbara Nagisa Ishiura

1 はじめに

産業機器, ロボット, 洗濯機や冷蔵庫等の家庭用電気製品, 冷却ファンやハードディスクドライブを内蔵した PC 等, モータは幅広い製品で使用されている. また, 近年では, 内燃機関を動力源としてきた自動車, 建設機械あるいはドローン等の輸送機器の電動化が急速に進みつつある. その一方, モータは日本国内の総電力消費量の 57.3% を消費しており [1], 省エネルギー化を達成するために, さらなるモータ制御の高度化が必要とされている.

モータのフィードバック制御では, センサから取得した回転速度や位置 (角度) 情報を元に PWM (pulse width modulation) 等でモータの駆動電圧を調整し, その回転速度やトルクを目標値に近付ける. これに対し, センサ情報を用いないセンサレス方式では, 制御系側でモータの物理的なモデルをシミュレーションすることによって速度やトルクを推定し, フィードバック制御を行う. モータのふるまいのシミュレーションには浮動小数点演算を多用するが、マイコン搭載の浮動小数点演算器の能力が限られていることもあり, 必要とされる制御周期を実現できない状況が生じつつある [2].

モータ制御の高度化を達成するアプローチの一つに, FPGA (field programmable gate array) 等を用いて制御処理をハードウェア化する方法が考えられる. FPGA が搭載している多数の乗算器を再構成することによって浮動小数点演算を並列に実行し, より短い制御周期でモータのふるまいを推定し, 制御を高度化することが可能になる.

C 言語等で書かれた制御プログラムを FPGA に実装可能なハードウェア回路に変換することは, 高位合成技術 [3] により比較的容易になった. しかし, 一般的な高位合成技術では, マイコンのアセンブリ言語で記述された割り込み処理プログラムを合成することはできず, タイマやセンサからの割り込みに基づく制御プログラムをそのままハードウェア化することはできない.

本稿では, 割り込み処理を含むプログラムを合成可能なバイナリ合成系 ACAP [4] を用いることにより, 割り込み駆動のモータ制御をハードウェア実装する手法を提案する. タイマ割り込みにより駆動されるブラシ付き DC モータのセンサレス制御を C 言語と MIPS のアセンブリ言語により記述し, これを ACAP で FPGA 上の回路に合成することにより, 約 21 µs の周期で PID 制御が行えることを確認した.

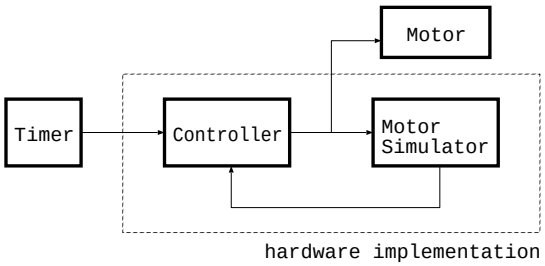

図 1 モータのセンサレス制御

2 モータの制御とモータの動作モデルのバイナリ合成

2.1 モータのセンサレス制御

本稿では, 図 1 [5] の破線内部の “モータのシミュレーション” と “PID 制御” からなるセンサレス制御のソフトウェア処理のプログラムから, マイコンと同様にタイマからの割り込み信号に同期して動作するハードウェア回路を合成する.

2.2 伝達関数によるモータモデル

図 2 [5] は, ブラシ付き DC モータの伝達関数をブロック線図で表したものであり, 印加電圧 V_a と負荷トルク T_L から回転角速度 ω を求めることができる. 図中 s はラプラス演算子, T_e $(= L_a/R_a)$ は電気的時定数, R_a は巻線抵抗, K_T はモータ定数, D は粘性制動係数, J は慣性モーメント, K_E は逆起電力定数である.

このモータの物理モデルを常微分方程式で記述し, 数値解析を行うことにより回転速度 ω を求めることができる. 文献 [6] では, 同様のモデルを記述した C プログラムを高位合成によって FPGA 上の回路に合成することにより, モータの高速なシミュレーションが可能であることを示している.

† 関西学院大学, Kwansei Gakuin University
‡ 京都高度技術研究所, ASTEM RI

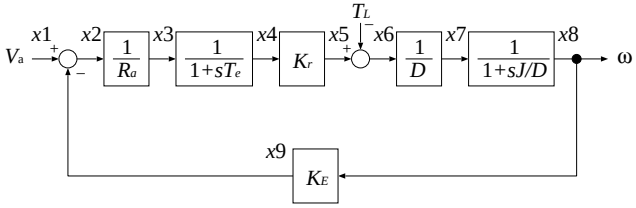

図 2　ブラシ付き DC モータの伝達関数のブロック線図 [5]

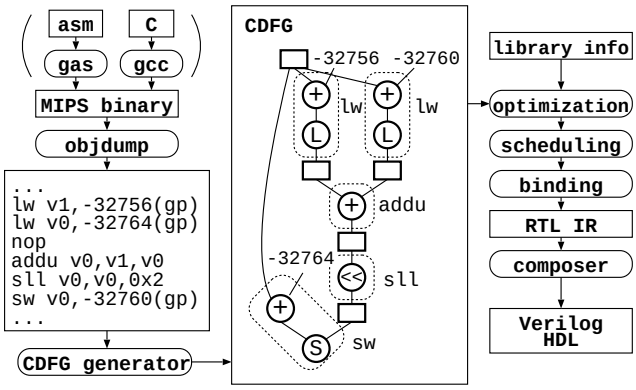

図 3　ACAP での高位合成の処理の流れ [4]

2.3　高位合成とバイナリ合成

高位合成は, C 言語等による動作記述からハードウェアの設計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はバイナリ合成 [7] 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, ポインタや構造体等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.

バイナリ合成システム 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. ACAP の処理の流れを図 3 に示す. ACAP への入力は, 既存の機械語プログラムであってもよいし, アセンブリ言語や C 言語によるプログラムから得られるものであってもよい. 機械語プログラムは CDFG (control dataflow graph) に変換された後, 最適化, スケジューリング, バインディングの処理を施して, Verilog HDL に変換される.

ACAP では, 割り込みハンドラを含むプログラムの高位合成が可能である [8]. 図 4 のように, プロセッサ及び割り込みハンドラを含むプログラムに等価なハードウェアを, 通常処理 (main) と割り込み処理 (int) を行う 2 つのモジュールとして合成する. 生成されるハードウェアの構成を図 5 に示す. CP0 は割り込み信号の処理を行う MIPS のコプロセッサである. main_module と int_module はそれぞれが状態機械を持っており, 独立に動作する.

3　モータ制御の記述

本稿では, 内部にモータの動作モデルを持ち, タイマ割り込みによって駆動されるモータ制御を, バイナリ合成によりハードウェア化する.

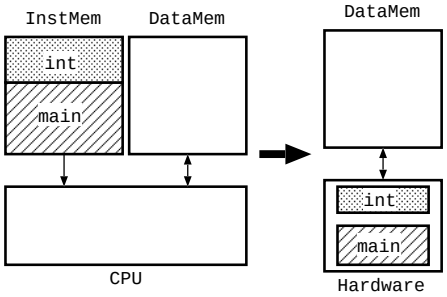

図 4　ACAP における割り込みハンドラを含むコードのハードウェア化 [8]

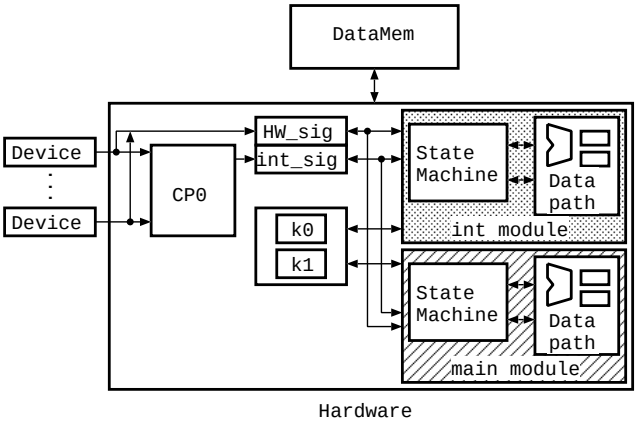

図 5　ACAP により合成されるハードウェアの構成 [8]

本稿のモータ制御には PID 制御を使用する. これはフィードバック制御の一種であり, 偏差, 偏差の積分, 偏差の微分をそれぞれ制御するパラメータである, 比例ゲイン K_P, 積分ゲイン K_I, 微分ゲイン K_D を調整することにより, 応答時間, 定常偏差, 振動, オーバーシュートを抑制する.

PID 制御とブラシ付き DC モータのシミュレータを含めたブロック線図は, 図 6 [5] のように定数と 3 個の 1 次遅れ要素で記述できる. 図中の $x1$ が制御されるモータへの印加電圧である. PID の各要素 (比例要素, 積分要素, 微分要素) はまとめて 1 つのブロック PID として表している.

本稿のモータ制御は, 周期的もしくは非同期の外部割り込みに基づいて行う. 割り込み信号が入ると, モータの回転速度と目標値を取得し, モータへの印加電圧を決定する. 割り込み時間の間隔を短くすれば, より高精度な制御を行えるが, 印加電圧の計算は次の割り込みまでに終える必要がある.

3.1　C 言語による記述

本稿のモータ制御を行うプログラムは図 7 のように記述できる [5]. 関数 `Pid_Control` は, 制御量 (現在の回転速度) `pState->speed` と目標値 (目標速度) `pInput->wCmd` からモータへの印加電圧を計算し, `pInput->wCmd` を修正する. 18 行目の関数 `Pid` は, 比例, 微分, 積分ゲインに基づき, 比例要素, 積分要素, 微分要素の処理を行う. 21 行目の関数 `DcLimit` は, 入力値が特定の範囲に含まれない

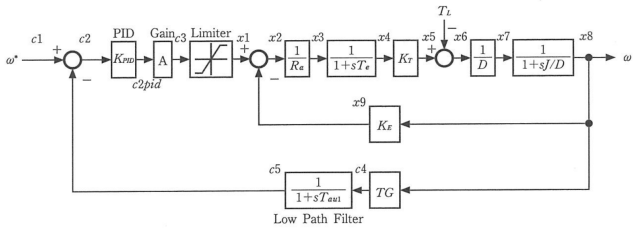

図 6　ブラシ付き DC モータの PID 制御の伝達関数のブロック線図 [5]

```
1  void PID_Control(
2      MotorInput *const pInput,
3      MotorState *const pState,
4      const float dt
5  )
6  {
7      static float c5 = 0.f;
8      static float work[2] = {};
9
10     const float KP = 1.0f,
11                 KI = 0.0f,
12                 KD = 0.03f;
13
14     const float GAIN  = 500.f;
15     const float c1    = pInput->wCmd;
16     const float c2    = c1 - c5;
17     const float c2pid =
18         Pid(c2, KP, KI, KD, work, dt);
19     const float c3    = c2pid * GAIN;
20
21     pInput->wCmd = DcLimit(c3);
22
23     motor(pInput, pState, dt);
24
25     const float TG   = 10.0f / 314.159f;
26       // 速度検査器設定 (3000/min : 10V)
27     const float c4   = pState->speed * TG;
28     const float TAU1 = 0.01f;
29     c5 = rk4(c4, c5, TAU1, dt);
30 }
31
32
33 float Pid(
34     const float xin,
35     const float kp,
36     const float ki,
37     const float kd,
38     float work[],
39     const float dt
40 )
41 {
42     const float a = xin;
43     const float b =
44       (xin + work[0])/2.0f * dt + work[1];
45     const float c =
46       (xin - work[0])/dt;
47
48     work[0] = xin;
49     work[1] = b;
50
51     const float xout =
52         a * kp + b * ki + c * kd;
53     return xout;
54 }
55
56
57 float DcLimit(const float input)
58 {
59     const float MAX_V  = 80.f;
60     const float MIN_V  = -80.f;
61
62     return
63           (MAX_V < input) ? MAX_V
64         : (input < MIN_V) ? MIN_V
65         :                   input;
66 }
```

図 7　モータの PID 制御 [5]

場合にこれを範囲内の値に修正するものである．23 行目の関数 Motor はモータシミュレーションの呼び出しであり，29 行目の関数 Rk4 はルンゲ・クッタ法による一次遅れ要素の近似計算である．

```
1  void Motor(
2      MotorInput *const pInput,
3      MotorState *const pState,
4      const float dt
5  )
6  {
7      static float x4 = 0.0f;
8      static float x8 = 0.0f;
9      static float x9 = 0.0f;
10
11     const float Ra   = 1.0f;
12       //電機子抵抗
13     const float La   = 0.008f;
14       //電機子インダクタンス
15     const float TAUe = La / Ra;
16     const float K    = 0.28f;
17       //モータ定数 (KE = kT = K)
18     const float J    = 0.005f;
19       //負荷を含む慣性モーメント
20     const float D    = 0.0002f;
21       //粘性制動係数
22     const float TAUj = J / D;
23
24     const float x1  = pInput->wCmd;
25     const float x2  = x1 - x9;
26     const float x3  = x2 / Ra;
27                 x4  = rk4(x3,x4,TAUe,dt);
28                  //電機子電流の計算
29     const float x5  = x4 * K;
30                  //モータ発生トルクの計算
31     const float x6  = x5 - pInput->
           loadTorque;
32                  //負荷トルクを印加
33     const float x7  = x6 / D;
34                 x8  = rk4(x7,x8,TAUj,dt);
35                  //回転角速度の計算
36                 x9  = x8 * K;
37     pState->speed   = x8;
38     pState->current = x4;
39 }
```

図 8　モータのシミュレーション [5]

モータのシミュレーションは，図 8 のように記述できる [5]．関数 Motor は，図 2 のブロック線図において，モータの各パラメータを，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，制御時間間隔 dt=500µs，回数 te/dt=2000 と設定したものである．4 次のルンゲ・クッタ法により一次遅れ要素の近似解を求める 関数 Rk4 は図 9 の通りである．

図 10 はメイン関数の記述であり，5 行目の関数 init_interrupt で割り込み処理の初期化を，6 行目の関数 register_exc_handler で割り込みハンドラの登録を行っている． なお，init_interrupt，register_exc_handler，および割り込み受け付けルーチンは，[8] と同様，インラインアセンブリを含む C プログラム及び MIPS アセンブリプログラムで実装している．

図 11 は割り込みハンドラの記述である．6 行目の関数 GetTime で現在時刻を取得し，16 行目で，前回の制御時刻からの経過時間，モータへの負荷トルクの値を制御関数 PID_Control に引き渡している．

4　合成結果

高位合成系 ACAP を用いて，3 章のプログラムを合成した．生成された Verilog HDL は論理合成ツール Xilinx ISE 14.7 によって，Xilinx Spartan 6 をターゲットに論理合成した．浮動小数点演算器に関しては，加減算器と

```
// 4 次のルンゲ・クッタ法による近似解
float Rk4(const float xin, const float
    xold, const float tau, const float dt)
{
    const float k1 =
      dt * (xin - xold) /tau;
    const float k2 =
      dt * (xin - (xold + k1 / 2.f)) /tau;
    const float k3 =
      dt * (xin - (xold + k2 / 2.f)) /tau;
    const float k4 =
      dt * (xin - (xold + k3)) /tau;

    const float xout =
      xold + (k1 + 2.f * k2 + 2.f * k3 +
          k4) / 6.f;
    return xout;
}
```

図 9　4 次のルンゲ・クッタ法による近似解計算 [5]

```
volatile int fContinue = 1;

int main()
{
    init_interrupt();
    register_exc_handler(
        EXC_Int0, TimerHandler
    );
    InitializeSim(); // 出力関連の初期化

    while ( fContinue ) /* VOID */;

    FinalizeSim();
    return 0;
}
```

図 10　割り込みハンドラの登録

```
void TimerHandler()
{
    // 前回の制御時刻
    static float time_l;
    // 現在の時刻
    const float time = GetTime();

    const float dt = time - time_l;
    time_l = time;

    MotorInput input = {
        5.0f,              // 速度指令
        GetCurrentLoad()   // 現在の負荷
    };

    PID_Control(&input, &state, dt);

    loopCount++;
}
```

図 11　割り込みハンドラ

乗算器は [6] で設計されたものを, 除算器と比較演算器は Xilinx Core Generator により生成したものを使用した.

実験結果を表 1 に示す. FF 数, LUT 数, 遅延, サイクル数, 周期は, それぞれフリップフロップ数, LUT 数, 遅延時間, 1 周期の制御に要するサイクル数と時間を表す. MIPS は文献 [9] の MIPS R3000 互換プロセッサであり, HW は本稿で合成したハードウェアである. プロセッサは浮動小数点演算器を持っておらず, 浮動小数点演算は soft float 実行時ライブラリにより行っている. 本稿のハードウェア実装では 1 周期の制御が 20.62 µs で行える.

表 1　論理合成およびシミュレーションの結果

	FF 数	LUT 数	遅延 [ns]	サイクル数 (周期 [µs])
MIPS [9]	1821	3788	16.078	23001 (369.8)
HW	2766	16617	23.676	871 (20.6)

5　むすび

本研究では, モータの伝達関数モデルを用いた割り込み駆動のセンサレス制御プログラムを, 高位合成系 ACAP を用いてハードウェア化した. 約 21 µs 間隔で割り込みをかけて PID を制御を行うことができた. 割り込みを活用したさらに高度な制御への適用が今後の課題である.

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, 元関西学院大学の田村真平氏に感謝致します. 本研究は一部 JSPS 科研費 60193265 の助成による.

参考文献

[1] “電力機器の消費電力量に関する現状と近未来の動向調査,” http://www.sicalliance.jp/science_data/bunken/fed-power-consume.pdf, 財団法人新機能素子研究開発協会 (Mar. 2009).

[2] 江崎雅康: FPGA マガジン no.7, pp. 4–7 “なぜ FPGA でモータを制御するのか,” CQ 出版社 (Oct. 2014).

[3] D. D. Gajski, N. D. Dutt, A. C-H Wu, and S. Y-L Lin: *High-Level Synthesis: Introduction to Chip and System Design*, Kluwer Academic Publishers (1992).

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

[5] 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).

[6] 竹林陽, 伊藤直也, 田村真平, 神原弘之, 石浦菜岐佐: “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,” 情処関西支部大会, A-02 (Sept. 2014).

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

[8] 伊藤直也, 石浦菜岐佐, 冨山宏之, 神原弘之: “割込みハンドラを独立したモジュールとして実装するバイナリ合成,” 信学技報, VLD2015-106 (Jan. 2016).

[9] 神原, 金城, 矢野, 戸田, 小柳: “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,” 情処関西支部大会, A-09 (Sept. 2009).