Erlang による組込みシステムの制御記述とその高位合成

東 香実 † 石浦 菜岐佐 † 竹林 陽 † 吉田 信明 ‡ 神原 弘之 ‡
Kagumi Azuma Nagisa Ishiura Hinata Takebayashi Nobuaki Yoshida Hiroyuki Kanbara

1 はじめに

組込みシステムの開発においては, 高機能化するシステムを, 定められた応答時間, サイズ, 消費電力等の制約下でいかに設計するかが課題となっている.

特に, 近年のネットワーク環境の普及やこれを利用した IoT (internet of things) サービスの展開に伴ない, 組込みシステムには単体での動作だけでなく, 他のシステムとの連携が求められるようになってきている. 組込みシステムの機能はこの点でも高度化しており, システムの仕様記述や効率的な設計手法が今後重要な課題になると考えられる. この課題に対する一つのアプローチとして, Erlang [1] 等の並行処理プロセスに基づいたメッセージ処理指向言語を用いてシステムの制御を記述する方法が考えられる. Erlang は, 近年では主として大量のメッセージをさばく Web サービスの実装に用いられているが [2], これを組込みシステムの実装に用いる試みも行われている [3].

その一方で, IoT サービスの展開には機器の一層の低消費電力化が必須である. また, 複雑な処理に対応する応答性能の確保, あるいは応答性能の飛躍的な向上は, 新たな応用を開発する上で重要な課題であり, システムのハードウェア化を視野に入れることが必要になってくる.

以上の背景を踏まえ, 我々の研究グループでは, Erlang による組込みシステムの制御の実装, およびその記述からハードウェアを自動生成する高位合成技術 [4] の研究を進めている.

2 Erlang

Erlang [1] は, 並行処理指向の関数型言語であり, 複数の並行プロセスと非同期のメッセージ通信によりシステムの動作を記述する. プロセスは非常に軽量であり, 多数のプロセスを生成して大量のメッセージを処理することができる. Erlang は, 交換機の実装言語として開発されたものであるが, 近年では主として Web サービスの実装に用いられている [2]. 一方で, 並行プロセスとメッセージ通信によってイベント処理が簡潔に記述できる点に着目し, Erlang を組込みシステムの実装に用いる試みも行われている [3].

Erlang のプログラムは, 仮想マシン BEAM のバイトコードにコンパイルされ, インタプリタで実行される. BEAM は 1024 個の 32 ビットレジスタを持つレジスタマシンであり, そのインタプリタは C 言語で実装され, Linux 等の Unix 環境で動作する.

3 Erlang による組込みシステム制御の記述

3.1 概要

本手法では Erlang のサブセットによりシステムの動作を記述する. 記述には複数の Erlang プロセスを用いるが, 全てのプロセスはシステムの起動時に生成し, 実行時に新たなプロセスの生成やプロセスの削除は行わないものとする. システムへの入出力は Erlang の「ポート」を介して行い, ポートはバイト系列の入出力を行うものとする. イベントに対する処理はメッセージの授受に基づく計算により表現する. なお, 現時点では, 使用するデータ型は 28 ビットの符号付き整数, およびリストとタプルに限定し, 関数型は扱わないものとする.

3.2 記述例

Erlang による簡単な動作記述の一例として,

- 進行方向を指示するボタン入力を受け付け,
- それを駆動系への制御信号に変換して出力する

というコントローラの記述を図 1 に示す. コントローラは図 2 のようなポートとプロセスでモデル化するものとする. `port0` がボタンからの入力を受信し, 制御信号を `port1` に出力する. ボタン入力データの駆動制御信号データへの変換はプロセス `proc1` が行う. プロセス `proc0` は, `port0` に入力があると, そのデータを `proc1` に送って制御信号に変換したものを受け取り, それを `port1` に出力する.

図 1 中, 4 行目の `start` がシステムを起動する関数であり, 5 行目でプロセス `proc1` を, 9 行目でプロセス `proc0` を生成している. `proc1` は本体 `loop1` (35–45 行目) を実行する. `proc0` は, 11 行目と 12 行目でそれぞれポート `Port0` と `Port1` を開き, 本体 `loop0` (20–33 行目) を実行する. `proc0` は `Port0` から制御の変換結果をメッセージとして受け取るとデータを整形して `proc1` に転送する (22–25 行目). また, `proc0` は `proc1` からメッセージを受け取ると, それを `Port1` に転送する (26–28 行目). それ以外のプロセスおよびポートからのメッセージは削除する. `proc1` は `proc0` からデータを受け取ると制御データの変換を行い, `proc0` に返送する. それ以外のプロセス及びポートからのメッセージは, そのまま `proc0` に転送する.

4 Erlang からの高位合成

4.1 概要

Erlang の実行環境は Raspberry Pi 等の小型の計算機にも実装されているが, Linux 上の Erlang 実行環境 (BEAM のインタープリタを含む) で実行されるため, システムとしては必ずしも軽量ではなく, 応答性能にも限界があると考えられる. Erlang のサブセットによるシステムの動作記述をハードウェア化できれば, これらの課題を解決できると期待できる.

本稿では, 前章で述べた Erlang のサブセットによるシステムの制御記述から, レジスタ転送レベルのハードウェア記述を合成する手法を提案する. 本手法では, Erlang の 1 つのプロセスを 1 つのハードウェアモジュールに合成する. プロセスはメッセージ通信を除いて他のプロセスの実行の影響を受けずに実行できる. また, スケジューラやプロセスの管理等の実行時環境のオーバヘッドもな

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

```
01: -module(roomba).
02: -export([start/0]).
03:
04: start() ->
05:   spawn(fun() ->
06:     register(proc1, self()),
07:     loop1(0, 0)
08:   end),
09:   spawn(fun() ->
10:     register(proc0, self()),
11:     Port0 = open_port({spawn,"stdbuf -i0 -o0 -e0 od -h -w8
              /dev/input/js0 | ./controller"}, [{packet, 2}]),
12:     Port1 = open_port({spawn, "./roomba"}, [{packet, 2}]),
13:     loop0(Port0, Port1)
14:   end).
15:
16: decode([Dt,Dh,Et,Eh]) ->
17:   {((Dh bsl 8) bor Dt), ((Eh bsl 8) bor Et)};
18: decode(X) -> X.
19:
20: loop0(Port0, Port1) ->
21:   receive
22:     {Port0, {data, Data}} ->
23:       Data2 = decode(Data),
24:       proc1 ! {proc0, data, Data2},
25:       loop0(Port0, Port1);
26:     {proc1, Data3} ->
27:       Port1 ! {proc0, {command, Data3}},
28:       loop0(Port0, Port1);
29:     {Port1, _} ->
30:       loop0(Port0, Port1);
31:     _ ->
32:       loop0(Port0, Port1)
33:   end.
34:
35: loop1(D, T) ->
36:   receive
37:     {proc0, data, Data} ->
38:       {Drive, Turn} = calc(Data, D, T),
39:       Cmd = encode(Drive, Turn),
40:       proc0 ! {proc1, Cmd},
41:       loop1(Drive, Turn);
42:     X ->
43:       proc0 ! X,
44:       loop1(D, T)
45:   end.
46:
47: calc({Para, X}, Drive, Turn) ->
48:   if
49:     X == 258  -> {Para, Turn};
50:     X == 1026 -> {Para, Turn};
51:     X == 2    -> {Drive, Para};
52:     X == 770  -> {Drive, Para};
53:     true      -> {0, 0}
54:   end.
55:
56: encode(Drive, Turn) ->
57:   if
58:     Drive =< 57343, Drive >= 32768 ->
59:       if
60:         Turn =< 57343, Turn >= 32768 ->
                {146, 0, 127, 0, 63};
61:         Turn =< 32767, Turn >= 12288 ->
                {146, 0, 63, 0, 127};
62:         true                          ->
                {146, 0, 127, 0, 127}
63:       end;
64:     Drive =< 32767, Drive >= 8192 ->
65:       if
66:         Turn =< 57343, Turn >= 32768 ->
                {146,255,127,255,63};
67:         Turn =< 32767, Turn >= 12288 ->
                {146,255,63,255,127};
68:         true                          ->
                {146,255,127,255,127}
69:       end;
70:     true ->
71:       if
72:         Turn =< 57343, Turn >= 32768 ->
                {146,255,127,0,127};
73:         Turn =< 32767, Turn >= 12288 ->
                {146,0,127,255,127};
74:         true                          ->
                {146,0,0,0,0}
75:       end
76:   end.
```

図 1　コントローラの記述例

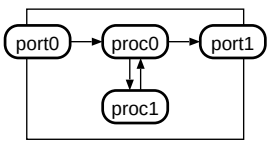

図 2　コントローラのプロセスとポート

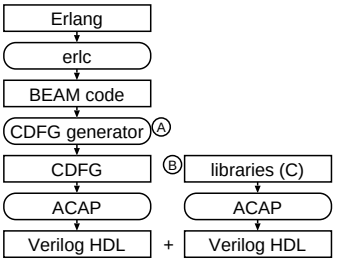

図 3　本手法の合成処理の流れ

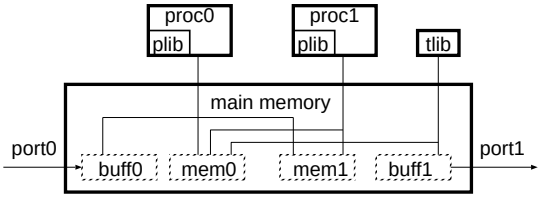

図 4　ハードウェアの構成

くなる．ただし本稿では, ハードウェアが使用するデータはすべて 1 つのメモリに置かれるハードウェア構成を想定する．

本稿の合成手法の流れを図 3 に示す．Erlang コンパイラ erlc で Erlang のプログラムから BEAM のアセンブリコードを生成し, ここから CDFG (control dataflow graph) を生成する．これを高位合成システム ACAP [5] のバックエンドに入力して Verilog HDL を得る．ACAP は C プログラムおよび MIPS R3000 の機械語プログラムを入力として, レジスタ転送レベルのハードウェアを合成する高位合成システムである．

ただし, BEAM の命令の中には, メッセージの送受信, ガーベジコレクション等, CDFG が大規模になるものが存在する．本稿では, これらの処理は, 別途独立したハードウェア (以下, ライブラリモジュールと呼ぶ) として実装し, プロセスを実行するハードウェアからサブルーチンのように呼び出せるようにする．ライブラリモジュールは, C 言語で実装された BEAM のインタプリタを縮約したものから ACAP で合成する．

図 2 の記述から合成されるハードウェアの構成は, 図 4 のようになる．proc0 と proc1 はプロセスの処理を行うモジュールであり, plib は各プロセスのライブラリモジュールである．tlib は出力ポート port1 のライブラリモジュールである．port0 から入力されるバイトシーケンスはメインメモリ内の buff0 の領域に格納され, port1 の出力は buff1 に格納される．mem0 と mem1 はそれぞれ proc0, proc1 の記憶領域である．proc0 から proc1 へのメッセージ送信は mem0 から mem1 へのコピーにより行い, proc0 から port1 への出力も mem0 から buff1 へのコピーにより行う．

4.2　実装

提案手法に基づく高位合成系のプロトタイプを開発中である．処理系は Linux 上で動作する．

BEAM アセンブリから CDFG を生成する処理系 (図 3 中の Ⓐ) は Perl5 で実装した．今回の実装では, CDFG 中の演算は ACAP の 32 ビット演算器を想定したものを生成した．ライブラリモジュール (図 3 中の Ⓑ) の C 言語記述は, Erlang OTP 18.1.3 中の BEAM インタープリタのソースコードから必要な部分を抽出して縮約す

ることにより得た.

現在, 図 1 の制御記述から論理合成可能な Verilog HDL 記述が生成可能である.

5 むすび

本稿では, Erlang のサブセットにより組込みシステムの制御を記述し, ここからハードウェアを合成する手法を提案した. 現在の実装で合成されるハードウェア構成では, プロセス数が増えた場合のメモリアクセスが処理速度のボトルネックになると考えられるため, 各プロセスにメモリを分散させるハードウェア構成を検討中である. 今後は, ガーベジコレクションによる応答遅延を解消する方法, BEAM を経由しないで Erlang から直接ハードウェアを合成する方法などについても研究を進める予定である.

謝辞

本研究に関して有益な御助言を頂いた立命館大学の冨山宏之教授, 元立命館大学の中谷嵩之氏に感謝いたします. また, 本研究に関してご協力, ご討議頂いた関西学院大学石浦研究室の諸氏に感謝いたします.

参考文献

[1] Joe Armstrong 著, 榊原一矢訳: “プログラミング Erlang” オーム社 (2008).

[2] 力武健次: “Erlang で学ぶ並行プログラミング,” Software Design, 2015 年 4 月号, pp. 124–129 (Apr. 2015).

[3] Brian Chamberlain: Using Erlang on the RaspberryPi to interact with the physical world (online), `http://www.slideshare.net/breakpointer/using-erlang-on-the-raspberrypi` (accessed 2016-02-04).

[4] 竹林陽, 石浦菜岐佐, 東香実, 吉田信明, 神原弘之: ”Erlang による組込みシステムの制御記述からの高位合成,” 信学技報, VLD2015-114 (Feb. 2016).

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