小規模制御系向けLMNtal処理系の設計と実装

矢島伸吾

7月27日

1　研究の目的

LMNtalはスケーラブルなソフトウェア基盤を目指している。本研究では、LMNtalが組込み機器における計算など、小規模な計算環境に応用可能であることを示すために、組込みシステム上で動作可能なLMNtal言語処理系の設計と実装を行なう。

組込み機器の特徴としては、

- メモリリソースが少ない
- 優先的に行ないたい処理がある
- 外部との入出力がある

などが挙げられる。まず、組込み機器はそのコスト的、電力的、スペース的制約から、メモリが少ないことが多い。そのため、メモリ使用量の少ない処理系設計を行なう必要がある。また、多くの組込み機器にはリアルタイム性が求められ、一定の時間内に処理を終える必要がある。LMNtalでそのような厳密なリアルタイム性を求めるのは難しいが、時間的制約を極力満たすためのアプローチとして、ある処理を優先して行なうなどの機構が必要となる。また、組込み機器においては様々な外部との入出力が行なわれるのが普通であり、それらの外部とのやり取りをLMNtalで表現する方法を考える必要がある。

LMNtalの特徴のひとつに、計算の簡潔なモデル化が可能であることが挙げられる。この研究により、組込みプログラミングにLMNtalを応用することができるようになれば、LMNtalを用いて組込み機器やそれらが形成するネットワーク上の計算を簡潔にモデル化し、容易に設計、開発することが可能となる。また、LMNtalは計算モデルであると同時に実用言語であるため、設計段階のモデル記述からプログラムとしての実装に落とす手間を最小限にすることができる。

本研究において対象とする組込み機器としては、ロボット制御及び組込みプログラミング教育用マイコンシステムであるeyebotを用いる。eyebotは

- アナログ、デジタル入力、サーボ、モータ等の制御や、カメラ、無線などの多彩な機能を持っている。
- 搭載メモリがROM512kB, RAM512kBと大容量であり、メモリ制約がゆるいため、処理系の実装を行ないやすい。
- 外部との入出力など多くの処理がAPIライブラリとして提供されており、ハードウェアの知識がなくてもC言語でプログラミング可能である。

という特徴をもっているため、LMNtal処理系の実装を行なう上で適している。eyebot上におけるLMNtal処理系の設計・実装を元として、より制約の厳しい組込み機器への応用にもつなげることができるだろう。

2　今までの成果

- eyebotの各種APIの使用方法、周辺ハードウェアとの接続方法などの調査およびそのドキュメント化 (wiki参照)
- LMNtalで動かしたいアプリケーション用のハードウェアの設計、製作
  - 温度計（温度を取得してLCDに表示。温度計は自作）
  - 防犯装置（PSDセンサ値の変化時にカメラから画像を取得し、PCに転送する。ソフト未実装。）
  - 目標に向かって進む車（カメラから画像を取得し、目印（赤いもの）を探して、その方向へ向かって進む。ソフト未実装。）

今までは、eyebot のハードウェアに関する調査を主に行なってきた。これにより、

- モータ、サーボ、エンコーダ、カメラ、無線、シリアル通信、アナログ入力、デジタル入出力の API 使用方法
- モータ、サーボ、エンコーダ、各種センサなどのハードウェア的知識および入手方法
- 回路設計技術の初歩と工作技術

を得ることができた。成果として、上記の３つのアプリケーションで用いるためのハードウェアを製作した。２輪で方向転換、前後移動が可能であり、車体前方のカメラによる画像取得、PSD センサによる距離取得が可能である。また、温度計を装着することで温度取得も可能となる。この１台に手を加えることなく、３つのサンプルプログラムを動かすことが可能である。また、エンコーダを用いた風向センサを用いることも可能となっている。

まだ上記３つのサンプルアプリケーションにおいて、ソフトウェアが完成しているのは温度計だけであるが、要素技術の調査は終了しているので、実装は可能である。(画像処理のみ若干の実験が必要かもしれない。) 今後は、これらのプログラムをまず C で実装し、それらのプログラムを LMNtal でどう表現したらよいかを考える。

3 LMNtal 処理系の設計方針

3.1 API について

API 群は大別して次のように分けられる。

- マルチタスク関連
- 初期化関数群
- 値の読み込み
- 値の書き込み

eyebot ではプリエンプティブ、協調型マルチタスク API が提供されているが、ハードウェア割込みは (おそらく) 使用できないため、マルチタスクの有効な利用法が思いつきづらい。LMNtal においては、「この膜は別タスクで動かし、並行動作させたい」という場合があるかもしれないが、セマフォを用いたロックなど、話がややこしくなるため当面実装は考えない。実行順序の制御は、優先度指定やプログラミングテクニックでカバーする方向で考えていきたい。

また、タイマーによる周期的割込みの機能も存在するが、これも同様に、付加価値的要素としての実装を、終盤において検討するにとどめる予定である。

これらの関数を除くと、残るほとんどの API は、ハードウェアの初期化および制御用 API である。LMNtal の処理系においては、初期化は決め撃ちで（必要となる、もしくは処理系の対応する全ての）ハンドラを取得してしまう。その後の制御は、atom の生成に対応して行なう予定である。

たとえば、温度計プログラムは C 言語では以下のように記述できる。

```
#include "eyebot.h"

int main()
{
  int tmp[10] = {0,0,0,0,0,0,0,0,0,0}
  int i = 0, j, ave;

  LCDMenu("", "", "", "END");

  while(KEYRead() != KEY4){
    /* reading temperature */
    tmp[i] = OSGetAD(2); OSWait(5);
    i = (i+1) % 10;

    /* calc average */
    ave = 0;
    for(j=0; j<10; j++) ave += tmp[j];
    ave /= 10;

    /* print */
    LCDSetPos(0,0);
    LCDPrintf("temperature:\n");
    LCDPutFloatS((float)ave / 25, 3, 1);
  }
  return 0;
}
```

この処理を LMNtal であえて逐次的に書いた場合、次のように書くことを考えている。

```
phase1
phase1 :- phase2(key(KEY4, key_read)), KEY4=64).
phase2(key(X,Y)) :- X = Y | .
phase2(key(X,Y)) :- X =/= Y | phase3(analog-in).
phase3(X) :- phase1, lcd(X / 25).
```

key_read, analog_in などの atom は外部入力を表す。ルール右辺でこれらの atom が生成された場合、即時

それらのアトムはその外部入力値 (Int) に置き換えられる。

lcd は外部出力を表す atom である。lcd(Int) は Int 値を LCD に出力する。外部出力 atom とそれに対する動作は、個別に規定する。例えばモーターへの出力は motor_a(50) などと表す。

ここにおいて、API 関連の atom の解釈、変換は、数値計算などのビルトイン処理より後に行なう。こうすることで、lcd(X/25) といった記述を可能にする。

3.2 ランタイム設計について

まだ仕様はできていないが、基本方針としてアトムとリンクを別々の双方向リストとして扱うことを考えている。アトムとリンクを一緒に扱おうとすると、リンク数によって atom の大きさが変わるため、GC 等の手間が大きくなる。アトムとリンクを別々にし、アトムは先頭リンクへのポインタを、リンクは次のリンクとリンク先アトムへのポインタを保持するようにし、未使用のアトムやリンクのノードは「空きアトムリスト」につなげておくことで、GC を行なわずに済ませることができる。

双方向リンクリストにするのは、atom の削除時にリストの任意個所の atom を削除しなければいけないからである。

膜のアトム管理方法は、膜ごとにアトム、リンクのリストを管理する方式を考えている。この場合、アトム、リンク領域はそれぞれ処理系全体で１つのまとまった領域として確保することができる。この他の方法として、膜ごとに領域を確保する方法があるが、これは膜の生成、消去にともなうメモリ管理が煩雑になるデメリットがある。

現状ではルール適用処理はスタックで管理しているが、スタックを別に用意するのはメモリ使用量が増える為避けたい。そこで、アトムにルール適用済みフラグを付加し、アトムのリストをリングバッファ状になめていき、ルール適用済みフラグが全アトムについていることを確認したら終了、という手順をとる。

また、ルール適用の高速化のために、通常のアトムと数値アトムを別々に管理する方式を取る。画像処理や姿勢制御などを考えると、float 値も扱えた方がいいため、数値領域は 32bit 確保する必要がある。そのため整数値もあわせて 32bit 用いる。この領域はリンク領域を流用するのが最も楽であると思われる。

マルチセットアトムはリンクありアトムと一緒に扱うが、リストを別々にすることで findatom の高速化が可能になるかもしれない。

この方針を元に、いかにメモリを少なくするかを考えていきたい。

リンクは 32bit で、

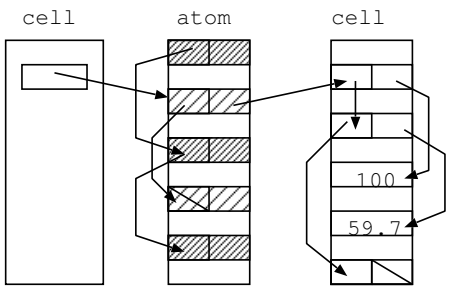
- 15bit: 次のリンクへのポインタ（最終リンクの時０）
- 15bit: アトムへのポインタ or 数値など（リンク領域を利用）へのポインタ or 文字を格納 (8bit)
- 2bit: リンク先種別（atom, int, float, string)

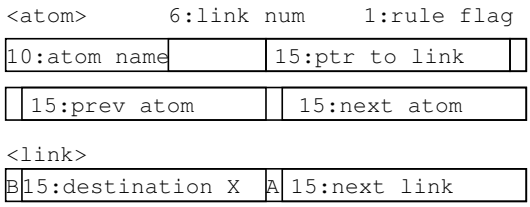
とあらわす。リンク領域 (32bit) は int, float 値, 文字格納領域としても使われる。

アトムは 32bit*2 で、

- 1bit: ルール適用フラグ
- 15bit: リンクへのポインタ
- 6bit: リンク個数
- 10bit: アトム名
- 15bit: 前のアトムへのポインタ
- 15bit: 次のアトムへのポインタ
- 2bit: reserved

といった実装を行なおうと思う。






```
A:numeral flag
  1:numeral(int or float)
  0:atom or string
B,if A=1:
  1:int 0:float
B,if A=0:
  1:atom 0:string
X if numeral:
  pointer to int/float
  (use link space)
X if atom:
  pointer to dest atom
X if string:
  top 8bit keeps character
```

膜は、

- 膜内のアトムのリンクリスト
- ルール

をもつ。ルール本体はシステム領域に最初に作成され、膜はルールへのポインタを持つ。また、システムは空きアトム、空きリンク領域をリストで管理している。

アトム、リンクへのポインタは、C のポインタではなく、アトム領域配列の添え字として表現する。こうすることで、必要な bit 数が減るだけでなく、プログラム中の扱いも簡単になる。各 15bit あれば、アトム 64bit*32k 個, リンク 32bit*32k 個で、計 384kbyte である。膜やシステムなどの領域を考えた場合、十分な量の領域が確保できると考える。

3.3 カメラについて

カメラ画像は、176*144 で 25k, 24bit なら 75kB である。82*62*24bit でも 15kB であり、それを Atom に落とし、Grid を形成した場合、1 画素につき Atom1,Link4 で 8byte 必要になるため、全体で 120KB 消費されてしまう。これより、高解像度での Atom 化は不可能、低解像度でも非現実的なメモリ使用量、処理速度になることが予想される。

よって、本処理系ではカメラ用領域をいくつかもっておき、

```
getpicture(1)
getpixel(1, 175, 143)
```

というように画像の撮影、画素の取得をおこない、画像処理は画素単位で値を取得して行なう方向で行こうと思う。

4 今後の方針

前述の 3 つのサンプルの C 言語コードを書き、実際に動かしてみる。そして、それを LMNtal でどう表現できるかを考える。

また、上記仕様をにつめ、ごく簡単なハンドコンパイルの中間コードを作成し、それが動く処理系を実装する。

現在の LMNtal 処理系仕様および中間コードの仕様を理解する。

組込み関係、リアルタイム関係などの参考文献のサーベイを行なう。