イベント駆動型並列プロセッサにおける制御ソフトウェアモデルの提案

林, 桐太 / HAYASHI, Touta

(出版者 / Publisher)
法政大学大学院情報科学研究科
(雑誌名 / Journal or Publication Title)
法政大学大学院紀要. 情報科学研究科編 / 法政大学大学院紀要. 情報科学研究科編
(巻 / Volume)
8
(開始ページ / Start Page)
195
(終了ページ / End Page)
200
(発行年 / Year)
2013-03
(URL)
https://doi.org/10.15002/00009877

イベント駆動型並列プロセッサにおける制御ソフトウェアモデルの提案

Software model for an autonomous vehicle using XMOS

林　桐太

Touta Hayashi
法政大学大学院 情報科学研究科
Email: 11t0026@stu.hosei.ac.jp

Abstract

Conventional microcomputers are often used in autonomous vehicles while suffering from the disadvantage of long complex codes containing unavoidable bugs. This paper describes how to solve complexity of software model on concurrent and event-driven processes with an embedded system using a new XMOS processor with exquisite ability for concurrent processes. In recent years, more and more software controllers are installed in many parts of a vehicle. Conventional microcomputers based on sequential- execution are not suitable for concurrent processes in a real-time system. So-called spaghetti-codes with insufficient interruption handling bring about serious problems. In contrast, an event- driven and multi-thread XMOS processor can accommodate simple and friendly codes using highly abstract modeling. A secure embedded system taking advantage of event-driven and multi-thread processors has been developed as a radio-controlled car with some sensors and simple codes on an XMOSX K-1 board. Successful results have been obtained with less efforts.

Index Terms—Concurrency, embedded system, event-driven, multi-thread, XMOS

1. はじめに

近年，自動車等の走行車両におけるソフトウェア制御の適用範囲は拡大している．これらの用途には組み込み向けマイコンが用いられる場合が多いが，構造的に逐次型である組み込み向けマイコンでは多数のセンサをリアルタイムに処理する事は難しく，ソフトウェアの肥大化，複雑化が避けられない．自律走行に必要な，単一のプロセッサで多数のセンサの入力を受付け，入力データを処理し，処理結果に応じて複数のデバイスへの出力指示をリアルタイムで行うような用途には，イベント駆動の並列型のプロセッサであるXMOSが適していると考えられる[1][2]．しかしながら，その適性に関わらずイベント駆動型並列プロセッサの利用状況は芳しくない．この原因としてはイベント駆動型並列プロセッサの構造上の特殊性から，ソフトウェア設計に従来型とは異なった手法・思想を必要とし，その手法が体系化されていない事が問題であると考えられる．本論文ではイベント駆動型並列プロセッサ向けソフトウェアモデルについて考察し，ファジィ理論[3][4][5][6]を取り入れた柔軟なソフトウェアモデルを提案し，自律走行車両への適用により検証を行う．

2. イベント駆動型並列プロセッサの特徴と現状

イベント駆動型並列プロセッサは，従来型組み込みプロセッサに比べて高い性能を持ち，並列実行におけるリアルタイム性能で他を大きく突き放すが，機器制御用としては従来型のマイコンを駆逐するに至らず，むしろARMなどのスマートフォンを出自とする汎用プロセッサに後追いされる形となっている．従来，それらの汎用プロセッサはコストが問題となり利用されてこなかったが，スマートフォン市場の活性化による量産効果と製造技術の微細化によりコストダウンがはかられた事を契機に組み込み市場への注力を始めている．また，画像処理などを含むより高度な処理・制御を求められる場合においてはPCを用いた制御が一般的となっているなど，イベント駆動型並列プロセッサは非常に厳しい曲面に立たされていると言える．

一方でUSB Audio 2.0インターフェイスデバイスにおいては公式にリファレンスデザイン及びリファレンスコードが用意された事が功を奏してかXMOSが実質的にシェア独占状態にあるなど，ニッチ市場においてはイベントドリブン型並列プロセッサは十分な競争力を持っている事も確認できる．このようにイベント駆動型並列プロセッサは，性能面では十分な選択肢になりうるものの，その設計において並列モデルが懸念となり積極的に採用されていない状況が見て取れる．

2.1. 提案手法の適用対象と検証

イベント駆動型並列プロセッサが真価を発揮する小規模の組み込みに対して検証を行なう．例としてXMOSを用いた自律走行車輌システム([Fig. 1])の制御プログラムについて検証する．

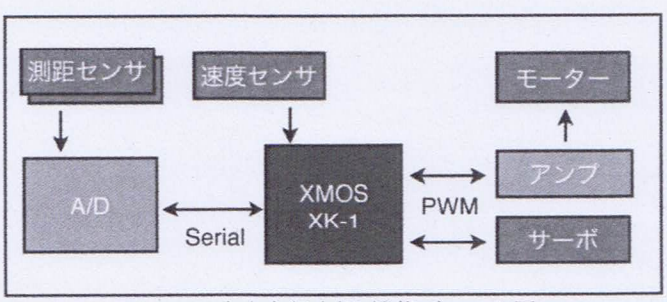


Fig. 1　自立走行車輌 機能ブロック図

Supervisor: Prof. Kenji Ohmori

3. 関連技術

3.1. XMOS

XMOSはXMOS社より提供されているイベント駆動型マルチスレッドプロセッサ及びそれを搭載した開発ボードのことを言う．特徴として，タイムスライシングによるハードウェアマルチスレッド，マルチコアと実行コアを意識しないメッセージ交換，抽象化されたI／Oインターフェイス，を持ち，ソフトウェア実装によるハードウェア機能の置換を特徴としている[7]．

XMOSプログラムはC言語および独自拡張のXC言語で記述が可能で，プログラム骨格はXC，アルゴリズム部分はC言語と書き分ける事ができる．またXMOSプロセッサの内部コアたるXCoreは強力なスイッチを備えたRISCプロセッサであり，高級言語たるXCによる印象とは裏腹に命令セットまで公開されており，必要に応じてアセンブラでの記述も行える．XC言語の機能の一部はC言語APIとしても提供されている．

3.2. CSP理論

CSP — Communicating Sequential Process[8][9]は，1978年にC. A. R. Hoareによって提唱されたプロセス代数理論及び仕様記述言語である． 独立したプロセス群がメッセージパッシングによって同期した相互通信を行なう事についての形式体系で，並行動作するプロセスの合成によるシステムのモデルについて並行性の観点で検証する事ができる．CSPに基づいたプログラム言語Occam[10]とOccamによる並列処理に特化したコンピュータTransputer[11]がある．

初期のものは単なる並列プログラミング言語であったが改良を重ねることでプロセス代数として成立し，その挙動について数学的に検証可能である．モデルの妥当性についての検証は手計算で行なう事もできるが，FDR2，PAT3などのツールを用いることで高速かつ確実に確認を行なう事ができる[12]．

3.3. ファジィ理論

ファジィ理論は，1965年に Zadeh によって提唱された「ファジィ集合 (Fuzzy sets)」[3] に 端を発する．このファジィ理論により，それまで明確に定義する事ができなかった集合や曖昧な集合，特に，人間の持つ言葉の意味の曖昧さを数学的に取り扱うことが可能となった．

ある対象が集合に属しているかを表現するにあたり，通常の集合であるクリスプ集合は {0, 1} でYES/NOの正負判定を取る．対してファジィ集合は[0, 1]，つまり0〜1の範囲で曖昧な表現を行なう事ができる．

ファジィ集合を用いる事で 「背が低い/高い」といったものに対して「少し低い」や「とても高い」といった程度情報を付加した表現を行なったり，「約30度」といったように境界の曖昧な数値指定を行なう事が可能となる．

3.3.1. ファジィ推論とファジィ制御

ファジィ理論の応用として，ファジィ推論，そしてファジィ制御がある．ファジィ推論とはファジィ値に関して言語化されたルールを組み合わせて結果を求めるもので，そのファジィ推論に用いて制御を行なうものがファジィ制御になる．

ファジィ推論は命題論理による三段論法（AならばB，CはAである，よってCはB）をファジィによって拡張し，程度条件（Cは「少し」Aである，よってCは「少し」B）を加えたものである．

ファジィ制御はフィードバック制御の一種で，「熱ければ水を入れる」ファジィ推論ルールによる制御操作により目的状態への遷移を行なう．古典制御論によるPID制御（偏差，積分，微分による制御）に比べ，非線形対象への対応，オーバーシュート性能で優れるとされる．また，ファジィによる曖昧さを内包することで職人の経験・勘などを再現した主観的制御を実現できるとされる．

4. イベント駆動型並列モデルの設計

イベント駆動型並列モデルの設計の難しさと，提案手法について述べる．

4.1. イベント駆動型並列モデル設計の難しさ

従来型組み込みプロセッサに比べてイベント駆動型の設計が難しいと先に述べた．その理由は複数の要因によるものだが，その一つに人間の思考が本来的にシングルタスクであるために並列に動作する仕組みへの想像力の限界があると考えられる．

逐次型ならばプロセスの流れを段階に区切り順を追って処理する事で，状況を分割して適宜処理する事ができるが，並列型では同時に動作する複数プロセスのそれぞれ個別の挙動を把握しなければならず，それらの相互作用も含めた組み合わせ問題により一度に考えるべきパターンが増大し，その設計を困難とする．

例として，用意された特定のコースを走行する車輌のプログラムについて考える．そのような限定状況であれば発生し得る状況は限定され，想定すべき状態のパターンを列挙する事は容易に行なえる．そして設計すべきが逐次型であれば「それらの列挙された状況を順にチェックし，判断される最適な行動を行なう」といった手順リストの作成によってモデル設計を行なう事ができる．出来上がったモデルは特定コースを前提としたものにはなってしまうが，徐々に想定範囲を拡張し手順を追加していく事で汎用化することも可能となる．

対して，イベント駆動型並列では，発生し得る状況を想定した上で，その中のどの状況をイベントとして定

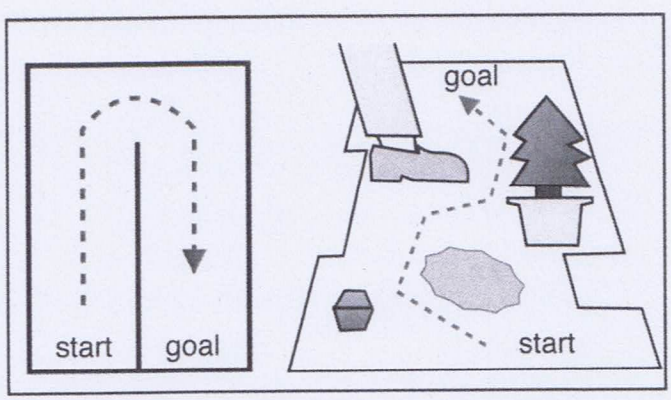


Fig. 2 特定コースと現実コースの差

義するかの判断を必要とし，さらに，そのイベントをどのようにモデルに対応付けるかを考えねばならない．モデルの関係性によってイベントの在り方も変化するなど，イベント定義とモデル設計は表裏一体であるために「玉子が先か鶏が先か」の状態を誘発される．あるいは並列性のないイベント駆動であれば，全ての状況を列挙した上でそれぞれへの対応を書くことでモデル設計を行なえるかもしれない．しかし，イベント駆動並列ではその後に並列化を目的としたイベントの分離・モデル分割が必要となり，場合によってはイベントの再定義が必要になるなど，結局に困難であることに変わりはない．

上記のようにイベント駆動並列は，特定条件下でのみ動作するモデルであってさえ適切なモデル設計を行なう事が難しく，より汎用的に機能するモデルではより一層に困難である．［Fig.2］で示すように現実世界で起こりうる星の数ほどのイベントの取捨選択を行い，必要な状況を網羅する事は事実上に不可能である．

4.2. ボトムアップ設計

手続き型指向ではトップダウン型でモデル設計を行なう事が多いが，前項で述べたようにイベント駆動並列には適さない．そのため提案手法ではボトムアップ型モデル設計を行なう．トップダウン型設計においては，まずモデルの全体像について『どうあるべきか』を定義し，そこからモデルを細分化していく．しかし，並列設計を多く含んだ全体像をイメージ・把握・設計する事は著しく困難であり，それがトップダウン型の実施を難しくしている．

そこで，提案手法では全体を見る事をせず，個々の機能にのみ着目してモデル化を行い，それらのモデルを接着する[13]ことによりを全体像を作り出すボトムアップ型設計を行なう．

ボトムアップ型の問題として一般に，個々の機能に注力を続けた結果に全体の挙動が破綻する危険性が挙げられるが，これはCSPによって回避する事ができる．また同じく問題となりやすい個々の偏りの積み重ねによる挙動の偏向の発生はファジィによる調整により回避する．

4.3. 機能モジュール設計

提案手法における具体的な設計の手順としては，入力，出力に着目しモデル化を行い，それらの間の不足を埋め繋ぐようにして順々にモデル化を行っていく．センサーなどからの外部入力，モーターなどへの外部出力，そのそれぞれにプロセスを割り当て，各々の入出力の取り得る値の範囲と，その意味に基づいて設計を行ない，それぞれを／それぞれが利用する機能を順に用意して行く事で全体を作り出す．

例として[Fig.3]のように測距センサーとモーターを台車に載せた，一次元上を移動する車輌のモデルについて考える．前述の方針に基づき，まずは入出力機能からモデル化を行うと最初に2つのモジュールができる．

入力は単純な測距センサーの機能モジュールであるので入力値はセンサー値，取りうる範囲はセンサーの入出力値，値の示すところはセンサー前方の障害物までの距離となる．この場合，出力すべきはセンサーの位置，向き，精度を考慮した特定向き方向の障害物への距離で

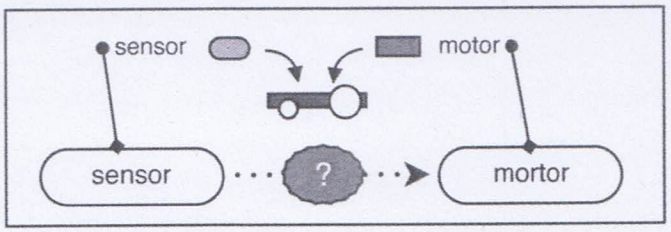


Fig.3　ボトムアップ型モデル設計

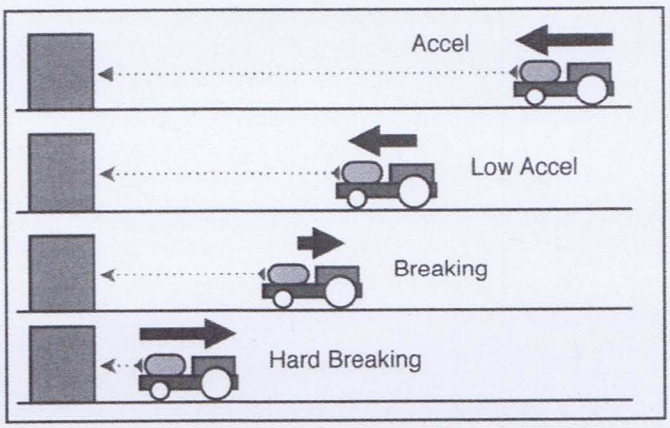


Fig.4　単純制御モデルの例

あり，このモジュールの機能は「センサー値を元に適切な値を出力する」と容易にモデル化できる．

出力側はモーター出力の機能モジュールとなるので，出力値はPWM波形，取りうる範囲はモータードライバの入力範囲，値の示すところはモーターへの出力値となる．こちらのモデルはモーターへの出力値を受け取って適切なパルス幅のPWM波形を出力するモデルとなる．

モーター出力モジュールのモデルが確定すると，それが必要とする値，モーターへの出力値を決定するモジュールが必要となり，ここで制御に関するモデルが発生する．制御については[Fig.4]のように障害物が近い場合は減速，なければ加速を行なうとすれば，モーターの出力は測距センサーの値を元に算出すれば良いこととなり，入出力の間を埋めるカタチになり，制御モデルが完成する．

このようにモデル化に際しては基本的になるべく小さい規模でモデル化を行い，それらを組み合わせる事で機能を実現していく．複数のセンサーでA/Dコンバータを共用している，実機材上のスレッド数の制限があるなど，幾つかの理由からモジュールの接着によるプロセスの生成を行なう事もある．

4.4. ファジィ化による処理値の抽象化

プログラム内で扱うデータ値について，センサーのraw値，raw値に処理を施したもの（fixed値），あるいは国際単位系(mm，m/h，rpmなど)への換算値などを用いる事が一般的であるが，提案手法ではこれに代わってファジィ化された抽象値を用いる．これにより高い生産性を確保する．各方式の特徴をまとめると以下になる．

・raw値：センサーから取得した生の値．
センサー出力がリニアでない場合などで，数値処理上に扱い辛い場合がある．

・fixed値： raw値を加工したもの．
逐次型設計ではこれで十分である．複数種類のセンサーがある場合に値の比較がし辛い．

・換算値： 国際単位系に変換した数値．
可読性が上がるが，換算・復号処理を繰り返す事になり，処理が増える．精度を保証・表現できない．

・抽象値： ファジィ化を行い，抽象化した値．
特定の着眼点でファジィ化するために値の意味が明確で，他方式での問題を解決できる．

逐次型設計ではセンサー値取得から出力処理までの一連の流れを逐次的に行なう仕組み上，連続したソースコード上で値と意味を共有できるためにあまり問題にならないが，イベント駆動型並列モデルでは並列化によってモジュール単位への処理の分割が行なわれてるためにモジュール間のデータの受け渡しが発生，受け渡し値の単位に関する問題が顕著となる．仮にセンサー取得と，取得値への処理判断，判断結果からの出力値決定，決定値の出力，と処理が分割されていた場合，それだけで値の受け渡しが3回発生し，それぞれに別の単位系を設定する事になる．どのような値を渡すのかの単位指定については仕様書において定義する事もできるが，それはモデル化の困難を増すだけである．

また組み込み分野ではif分岐の閾値などをソースコード中にハードコーディングする事が一般的だが，デバイス依存の数値に替わって意味を伴ったファジィ値を記述する事になりソースコードの可読性が向上する

4.5. ファジィ値のスカラー化

通常，ファジィ値はベクトル値を取る．設定されたファジィ集合を次元とし，対象の各々についての適合度をまとめたベクトルである．しかしながら，モジュール間で受け渡しを行なう都合上にスカラーの方が都合が良いので，スカラー化を行なう．

設定されるファジィ集合は主に2つに分けられる

・設定された集合が一次元上に配置されている場合

・設定された集合が二次元以上である場合

前者の一次元上に配される集合については，三角型メンバシップ関数においては以下のように評価できる．

ファジィ集合A,B,Cについて
三角型メンバシップ関数 HA(x)，HB(x)，HC(x)
評価値kA，kB，kCを設定した時
xのファジィ値 v は

$$v = (HA(x), HB(x), HC(x))$$

となり，これの評価値 s を

$$s = kA*A(x) + kB*B(x) + kC*C(x)$$

とする．

後者の二次以上の場合，各次元が各々に異なる意味を示しているので合成することはせず，各次元についての評価値を求める．

なお，この式において評価値として実値を設定すると非ファジィ化の操作と等価となる．

5. 制御における保証と可視化

自律走行車両などの自動制御機器では人の意志から離れて動作する事から，予期せぬ動作により怪我や人命に関わる事態が起きぬよう，その制御については厳密さが求められる事が多い．予期せぬ動作とは，主に期待にそぐわない動作，予定通りだが状況に合っていない動作に分別できる．

	値1	値2
raw値	2800	1500
fixed値	16	289
国際単位系(mm)	650	3000
fuzzy(遠い)	0.55	0.72

Fig.5 処理値の例

前者における厳密な制御とはその制御対象の内部に対して行なわれるものであり，特に並列システムにおいては協調動作の失敗からの停止など，主にデッドロックとして問題とされる．提案手法においてはCSPを用いて保証する．

後者である結果としての制御対象の挙動についての厳密さについては，ファジィを用いる事により曖昧となる事もあり，保証されない．しかしながらそもそも十分な情報が与えられることのない現実環境においては完璧な厳密さを求める事が本来的に不可能である．また実際上に必要とされていない場合も多く，あるいは敢えて必要としないという選択肢を取ることにより部品精度などの面において大きなメリットを享受する事もできる．

本論で利用する自立走行車両では構成部品として主に民生用，それもホビー用として販売されている物を用いている為，モーター，センサー等の部品精度は勿論として，そもそも寸法などから既に仕様が曖昧であり，緻密な制御とは縁遠い．寸法などにおいては現在の状態を測定する事はできるが，実測で得た値は現在値であること以上を保証できるものではなく，センサー等では非リニア値を取る場合もある．

ファジー制御を用いるメリットの一つはそのような曖昧な環境においても適切なメンバシップ関数を設定する事で許容値に収める事ができる事である．

5.1. CSPによる基礎モデル

制御システムを構成するプロセスは入力，演算，出力の三種類に分類される．センサーなどの外部機器からの入力を処理するプロセス，それを元に演算を行なうプロセス，そして演算結果を出力するプロセスである．

これらをCSPでモデル化すると以下のようになる

```
In = Read?x → TIMER?t → In
     [] Res!x → In

Exe = Ask?z → Res!z → Exe

Out = TIMER?t → Write!y→ Out
     [] Ask?y → Out
```

実際に利用する場合はrenamingを用いて必要要素を置き換え，繋げ並列化を行なう．一つのセンサー，一つのモーターから構成される最低構成の例をあげると以下のようになる［Fig.6］.

```
In[[ Read<-SensorOut, Res<-Ch1 ]]
      ||
Exe[[ Ask<-Ch1,Res<-Ch2 ]]
      ||
Out[[ Ask<-Ch2, Write<-MotorIn ]]
```

また複数センサーからの情報を処理するモデルは以下のようになる．

```
Exe2 = AskA?z → AskB?z → Res!z → Exe2

Exe3 = AskA?z → AskB?z → AskC?z
       → Res!z → Exe3
```

実際の制御システムの殆どがこれらの組み合わせにより表現可能であり，モデル検証を行う事ができる．

5.2. PAT3によるモデル検証

続いて先のモデルについてPAT3による検証を行なう．

先のモデルにセンサーとモーターを模したプロセスを追加し，renamingを反映させ，一部を厳格化したものを［Fig.7］に示す．

センサーは適当な信号を出力，モーターは入力だけを行なう事とした．

```
Sensor = SensorOut!1 → Sensor

Motor = MotorIn?x → Motor
```

加えてdeadlockfree，divergencefreeのassertionを宣言しておくことで，デッドロックフリー，ライブロックフリーに関する検証を行なう事ができる．

［Fig.8］のトレース図でわかるようにこのように簡略化したモデルにおいても相当数の状態遷移が発生している．プロセス代数を用いることなくこの一つ一つを取りこぼしなく確認する事の困難さが並列コンピューテイングにおけるバグ発見の難しさとモデル検証の意義の証明となっている．

5.3. メンバシップ関数による調整

モデル各所で行なわれるファジィ化は，各所に埋め込まれたメンバシップ関数によって行なわれる．ゆえに，これらのメンバシップ関数を調整する事により全体の挙動を指定・調整する事ができる．

メンバシップ関数の調整箇所としては，ファジィ分割数，分割位置，メンバシップ関数の形状，スカラー化／非ファジィ化時に用いる評価値が挙げられる．

基本的には評価値の適正化と分割位置の移動が調整の中心となるが，それ以外の項目も調整に用いる事ができる．分割数，位置については本来モデル定義段階で定義する項目だが，モジュール内部でのみ用いられ，モジュール外に出す時にはスカラー化の過程を経るため増減させても問題ない．メンバシップ関数の形状は処理の簡略化の為に三角型を採用しているが，必要に応じて釣り鐘型，台形型などを用いる事もできる．

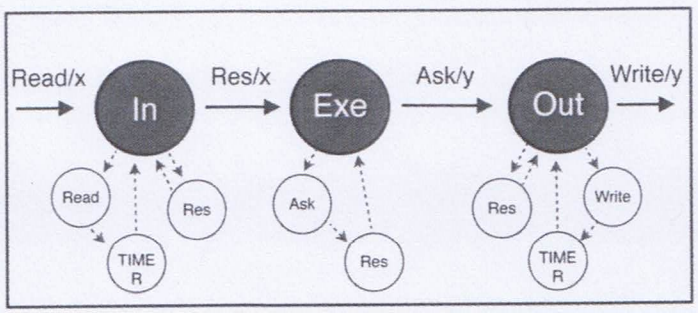


Fig.6　CSP基礎モデル概念図

```
1   channel Ch[3] 0;
2   channel SensorOut 0;
3   channel MotorIn 0;
4   channel TIMER1 0;
5   channel TIMER2 0;
6
7   In = SensorOut?x -> TIMER1?t -> In_(x);
8   In_(x) = SensorOut?x -> TIMER1?t -> In_(x)  [] Ch[1]!x -> In_(x);
9   Exe = Ch[1]?z -> Ch[2]!z -> Exe;
10  Out = Ch[2]?y -> Out_(y);
11  Out_(y) = TIMER2?t -> MotorIn!y-> Out_(y) [] Ch[2]?y -> Out_(y);
12  Sensor = SensorOut!1 -> Sensor;
13  Motor = MotorIn?x -> Motor;
14  TimerA = TIMER1!1 -> TimerA;
15  TimerB = TIMER2!1 -> TimerB;
16
17  SYSTEM = In || Exe || Out || Sensor || Motor || TimerA || TimerB;
18
19  #assert SYSTEM() deadlockfree;
20  #assert SYSTEM() divergencefree;
21  #assert SYSTEM() deterministic;
```

Fig.7　PAT3検証コード

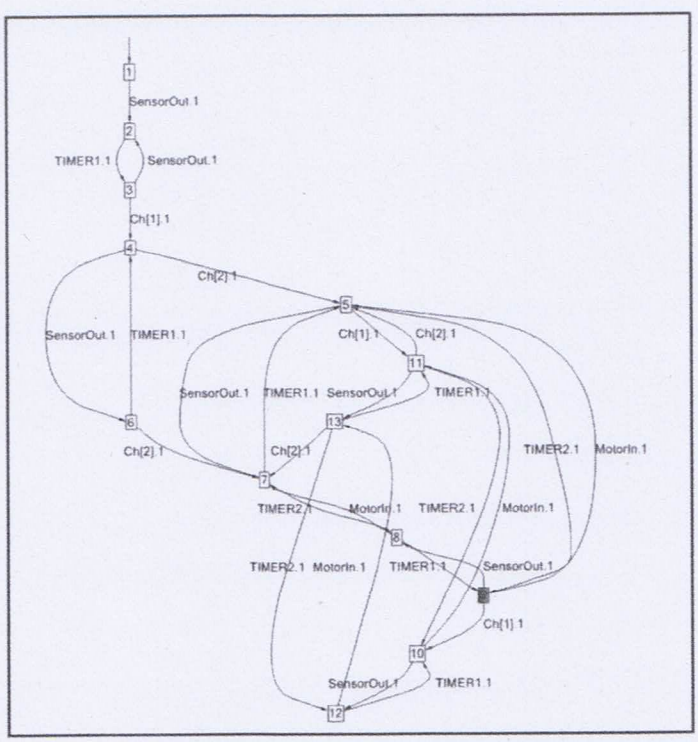


Fig.8　PAT3トレース結果

各ファジィ値はモデル化の時点でその値の意味するところが明確であるため，メンバシップ関数の調整に当たってはその目的にそって最適に設定するだけでよい．

5.4. キャリブレーション

メンバシップ関数による調整の一環として，センサーなどのキャリブレーションを行なう事ができる．

最小値，最大値と途中の何点かをファジィ集合として指定，それぞれに測定した実値を利用する事によりセンサー毎の個体差などの公称値からのズレと実際の値を特別な工夫なく擦り合せる事ができる．また，測距センサーなどではセンサー前方に障害物を設置した状態でのセンサー値を拾う事により，「近い」「少し近い」「遠い」といったファジィ集合に対してそれぞれ距離を直感的に設定する事ができる．目測での主観距離のを直接に行なうため，単位に影響される事がない．このように具体的な数値ではなく，主観による概念距離を指定することで，数字が概念に先行する事なく設計したモデルに忠実な実装を行なう事を可能とする．

5.5. 可視化

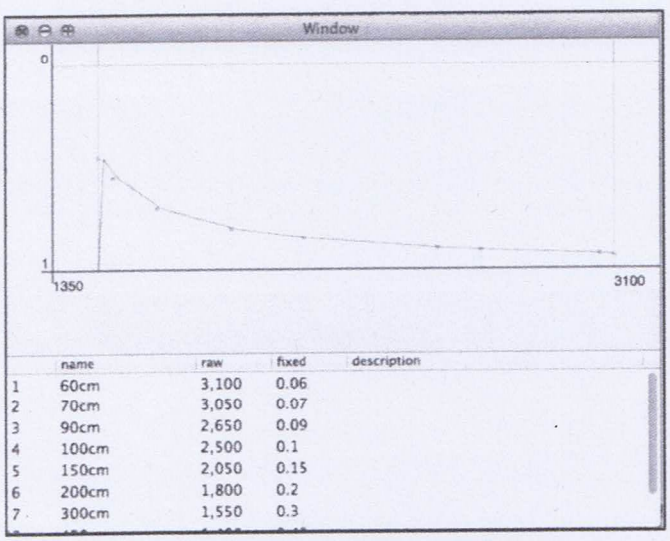
ファジィ理論の問題の一つとしてその調整，パラメータの適正化が難しい事が挙げられる．依存関係を持つ複数のパラメータを調整し，全域に渡って適切な最終出力を得る事が難しいのは自明であろう．

提案手法の制御精度はファジィに依存している為，適切なパラメータ設定は必須となる．この問題解決の為に可視化・調整のツールを開発した．

開発ツールではメンバシップ関数のグラフによる可視化とその場での変更を行い（[Fig. 9]），またその調整値をXCファイルとして書き出し，利用できる．

6. おわりに

以上のように，イベントドリブン型並列プロセッサにおける制御ソフトウェアモデル設計手法の提案を行ない，CSPによるモデル検証によるモデル妥当性の獲得とファジィを用いた評価値計算による高い開発効率を得る事ができた．



	name	raw	fixed	description
1	60cm	3,100	0.06	
2	70cm	3,050	0.07	
3	90cm	2,650	0.09	
4	100cm	2,500	0.1	
5	150cm	2,050	0.15	
6	200cm	1,800	0.2	
7	300cm	1,550	0.3	

Fig. 9　メンバシップ関数の可視化

また逐次型アーキテクチャの延長としての並行スレッド向けの設計手法ではなく，イベントドリブン型並列の特性に合った手法を用いることにより従来型アーキテクチャとは異なったメリットを享受することができる事を示した．

イベントドリブン型並列プロセッサの持つ潜在的ポテンシャルは非常に高く，様々な用途で活躍できると考える．今後の利用範囲の拡大に期待したい．

文献

[1] G. Martins, A. Moses, M. Rutherford and K. Valavanis, Enabling Intelligent Unmanned Vehicles Through XMOS Technology, The Journal of Defense Modeling and Simulation 01/2011; 9(1):71-82, 2011.

[2] Hayashi, T.; Ohmori, K. , "An autonomous vehicle using a multi-thread and event-driven processor". 13th International Symposium on Integrated Circuits (ISIC), 2011.

[3] L.A.Zadeh : "Fuzzy sets", Information Control, Vol.8, pp.228-353, 1965.

[4] 菅野道夫：『ファジイ制御』 日刊工業新聞社, 1988.

[5] 田中一男：『アドバンスドファジイ制御』 共立出版社, 1994.

[6] 浅野目哲也："鉄道車両の滑走防止制御における多段ファジィ推論に関する研究", 筑波大学大学院システム情報工学研究科修士論文, 2006.

[7] Douglas R.Watt, Programming XC on XMOS Devices, XMOS Limited, 2009.

[8] C.A.R. Hoare, Communicating Sequential Processes, 21(8): pp.667-677, 1978.

[9] D. May, Communicating process architecture for multicores. The 30th Communicating Process Architectures Conference, 2132. 2007.

[10] INMOS, occam Programming Manual. Prentice-Hall, 1984

[11] A. Kent and J.G. Williams, Encyclopedia of Computer , Science and Technology, Dekker, 1998.

[12] Ryo Mizutani; Ohmori, K. : "A Design and Implementation Method for Embedded Systems using Communicating Sequential Processes with an Event-Driven and Multi-Thread Processor", International Conference on Cyberworlds, 2012.

[13] K. Ohmori, and T. L. Kunii, Designing and modeling cyberworlds using the incrementally modular abstraction hierarchy based on homotopy theory. The Visual Computer, 26(5):297309, 2010.