M-84

ユビキタスコンピューティングのための入出力制御デバイスのソフトウェアアーキテクチャ

A Software Architecture of I/O Control Devices for Ubiquitous Computing

義久 智樹 *1　塚本 昌彦 *1　坂根 裕 *2　寺田 努 *3
Tomoki Yoshihisa　Masahiko Tsukamoto　Yutaka Sakane　Tsutomu Terada
岸野 泰恵 *1　早川 敬介 *4　柏谷 篤 *4　西尾 章治郎 *1
Yasue Kishino　Keisuke Hayakawa　Atsushi Kashitani　Shojiro Nishio

1. はじめに

いつでもどこでもコンピュータにアクセスできるユビキタスコンピューティング環境を実現するため，筆者らの研究グループでは，ルール形式で動作を記述する入出力制御バイスを用いたユビキタスコンピューティングを提案し，入出力制御デバイス（ユビキタスチップ）の試作を行っている [1, 2].

本稿では，試作したユビキタスチップのソフトウェアーキテクチャについて述べる．ユビキタスチップは記憶容量が乏しく，処理は逐次的であるため，ルール処理のブロック化などイベント駆動システムを実現するための様々な工夫が必要になる．

2. ユビキタスチップの仕様

ユビキタスチップは動作記述言語としてイベント駆動型言語である ECA ルールを用いる．ECA ルールとは，あるイベント（E）が起こり，特定のコンディション（C）を満たしているとき，そのアクション（A）を実行する，という動作記述言語である．ECA ルールを用いることで，例えば「入力状態が ON になったときに出力状態を ON にする」といったサービスが記述できる．ユビキタスチップは，ビット列に変換された ECA ルールを自身のルールベースに格納する．ECA ルールは 1 ブロック（2 バイト）単位で記述され，イベントおよびコンディションを記述する実行条件記述部と，アクションを記述する実行動作記述部で構成される [3].

ユビキタスチップを構成するハードウェアは小型で省資源の入出力制御デバイスである．一般に，小型で省資源の入出力制御デバイスには次のような問題があり，ECA ルールのエンジンを実現するためにはそれらの問題を解決する必要がある．

- 逐次処理
  逐次的な処理しか行えず，イベントが起こったことをトリガとして ECA ルールを処理するイベント駆動型機構を擬似的に実現する必要がある．
- 少ない記憶容量
  記憶容量が少ないため，ECA ルールを小さなブロックに分割し，1 ブロックずつ処理する必要がある．
- 一定間隔の割り込み処理
  あらかじめ定められた一定間隔の割り込み処理しか行えないため，任意の間隔の割り込み処理や複数タイマの同時利用を可能にするためには，タイマ処理機構を新たに実現する必要がある．

*1: 大阪大学大学院 情報科学研究科，Graduate School of Information Science and Technology, Osaka University
*2: 大阪大学大学院 工学研究科，Graduate School of Engineering, Osaka University
*3: 大阪大学サイバーメディアセンター，Cybermedia Center, Osaka University
*4: NEC インターネットシステム研究所，Internet Systems Research Laboratories, NEC Corporation

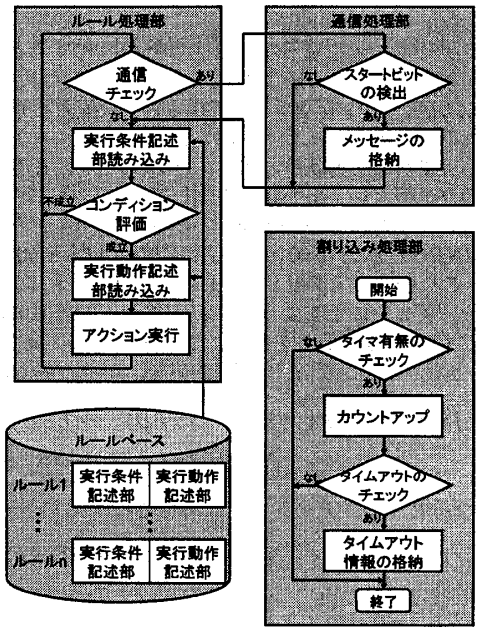

図 1: 想定するユビキタスチップのブロック図

3. ソフトウェアアーキテクチャ

ユビキタスチップにおける処理のブロック図を図 1 に示す．少ない記憶容量で，逐次的なルール処理が行える様に，ルール処理部はルールベースから実行条件記述部のみ読み込む．イベントおよびコンディションが満たされている場合には，実行動作記述部を読み込み，アクションを実行する．満たされていない場合には，続けて次の ECA ルールの実行条件記述部を読み出す．n 個の ECA ルールをすべてチェックすると，再び初めの ECA ルールからチェックする．通信チェック，通信処理部，割り込み処理部については後述する．

このような枠組みの中で，文献 [2] で示されている，ユビキタスチップに求められる機能の実現方法について述べる．

3.1 入出力制御

ECA ルールを用いてユビキタスチップの入出力制御を行うには，コンディションに ON/OFF の入力状態を指定し，アクションに出力状態を指定する．さらに，マスクを用いることで，ある端子の入力状態のみ評価したり，ある端子の出力

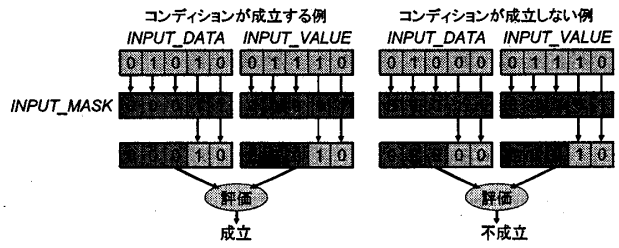

図 2: 入出力制御の例

状態は変更せず，他の端子の出力状態を ON にするといった制御を可能にする．*INPUT_DATA* を現在の入力状態，*INPUT_VALUE* をコンディションで指定された入力状態とする．それぞれの変数の $i\ (=1,\cdots,m)$ ビットめは，m 個ある入力端子の i 番目の入力状態とする．*INPUT_MASK* を入力端子のマスク情報とし，出力端子に関しても同じ様に定義する．例えば，*INPUT_MASK*=000011 の場合に，図 2 左に示すように *INPUT_DATA*=01010，*INPUT_VALUE*=01110 とすると，これらはマスク後 000011 となり，コンディションが成立する．図 2 右に示すように *INPUT_DATA*=01000，*INPUT_VALUE*=01110 とすると，これらはマスク後 000000，000010 となり，コンディションは成立しない．すなわち，

$$(INPUT\_DATA \ \text{XOR} \ INPUT\_VALUE) \ \text{AND} \ INPUT\_MASK$$

が 0 になると，コンディションは満たされ，出力端子を

$$(OUTPUT\_DATA \ \text{AND} \ \overline{OUTPUT\_MASK})\text{OR} \ (OUTPUT\_VALUE \ \text{AND} \ OUTPUT\_MASK)$$

で与えられる状態に変更する．

3.2 メッセージの送受信

メッセージはシリアル通信で送受信するため，スタートビット検出に失敗すると通信エラーが発生する．そのため，図 1 左に示す様に，1 個のルールをチェックする度にスタートビットを検出するための通信チェックを行う．スタートビットが検出されると，メッセージをメッセージ専用レジスタに格納する．受信したメッセージは RECEIVE イベントでコンディションの評価が行われ，評価後，レジスタから削除される．この様に，メッセージを受信してから RECEIVE イベントが評価されるまでの遅延が生じるが，すべてのルールをチェックする時間は数ミリ秒であるため，この程度の遅延は問題にならないと考えられる．

3.3 タイマ

想定するユビキタスチップでは，あらかじめ定められた一定間隔の割り込み処理しか行えないため，任意の間隔のタイマを実現することは困難である．そこで，タイマの間隔を割り込み処理間隔の整数倍に固定し，カウンタを用いて任意の間隔のタイマを実現する．割り込み処理でカウンタの値を 1 ずつ増やし，あらかじめ設定した値になると，タイムアウトしたこと示す情報をレジスタに格納する．例えば，20 ミリ秒間隔で割り込み処理が行われる場合に 100 ミリ秒間隔のタイマを発生させるには，カウンタが 50 になるとタイムアウトとみなすように設定すればよい．複数のカウンタを用いることで，間隔の異なる複数のタイマを同時に利用できる．タイムアウトしたタイマは TIMER イベントで評価される．

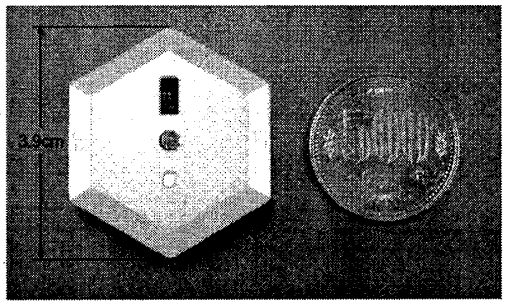

図 3: 試作ユビキタスチップ

3.4 ルールの追加，削除

文献 [3] に示すように，ルールの追加，削除のコマンドはシリアルポートを通して送信される．ルールの追加コマンドを通信処理部が受信すると，そのコマンドに続いて送信されるルールをルールデータベースに追加し，ルールの削除コマンドを通信処理部が受信すると，ルールデータベースの ECA ルールを削除する．

4. 実装

第 3 章で述べたソフトウェアアーキテクチャをアセンブラを用いて実現し，Microchip 社の PIC（Peripheral Interface Controller）に実装した．ルールベースとしては，PIC に内蔵されている不揮発性メモリを使用した．入出力装置の接続が容易になるようなケースを作成し，PIC および周辺回路を内蔵した（図 3）．通信エラーが頻発するという問題があったが，通信レートを 2400 bps に下げることで解決できた．

5. 最後に

本稿では，ルール形式で動作を記述する入出力制御デバイス（ユビキタスチップ）のソフトウェアアーキテクチャについて説明した．記憶容量が乏しく，処理が逐次的な場合には，ルールをブロック化して処理することで，容易にイベント駆動システムを実現できる．また，シリアル通信を用いてメッセージを送受信できるため，ユビキタスチップ同士の連携動作や PC との連携動作が可能になる [4]．

今後は PIC 以外の入出力制御デバイスへの実装や，アナログ入出力処理への対応など，より高性能なユビキタスチップを実現する予定である．

参考文献

[1] 早川ほか: ユビキタスコンピューティングのための入出力制御デバイスのハードウェアアーキテクチャ，FIT2002 論文集 (2002, 掲載予定).

[2] 塚本ほか: ユビキタスコンピューティングを実現するためのルールに基づく入出力制御デバイス，FIT2002 論文集 (2002, 掲載予定).

[3] 寺田ほか: ユビキタスコンピューティングのための入出力制御デバイスの動作記述言語，FIT2002 論文集 (2002, 掲載予定).

[4] 岸野ほか: ユビキタスコンピューティングのための入出力制御デバイスの PC 統合環境，FIT2002 論文集 (2002, 掲載予定).