計算機設計言語による複数装置の記述*

萩原　宏** 黒住祥祐***

Abstract

The multi-processor system of computer is useful to promote the speed and reliability of computer. One of the most important problem of the multi-system is the communications between computer units. In this paper, we deal the communications between units.

First, the unit which is the same as the sequential machine of Moore model is defined.

Next, the various communications between these units are classified and described in the design language of computer (T-language) which we proposed lately.

1. まえがき

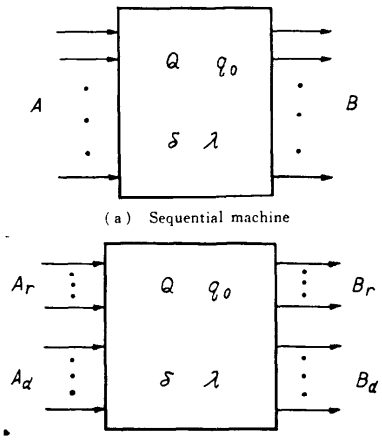
計算機の高速性および信頼性の向上の手段として，複数の処理装置による多重処理システムがある．多重処理システムの評価は，個々の処理装置の性能にも関係するが，むしろ，処理装置間の相互作用の効率に大きく左右される．本文はこのような装置間の通信に注目し，順序回路と同等の装置を定義し，次に装置間の通信方式を検討する．さらに，筆者らが先に提案した計算機設計言語（T-言語）[1] により，これらの通信方式を記述する．

2. 装置の定義

T-言語は Moore 形の順序回路の記述に適する．多出力 Moore 形順序回路は $(Q, q_0, A, B, \delta, \lambda)$ で定義される．ここで，Q は状態の有限集合で，$q_0 \in Q$ は初期状態の組を表わす．A は入力記号の有限集合で，a 次元の2進ベクトルの集合とする．B は出力記号の有限集合で，b 次元の2進ベクトルの集合である．δ および λ はそれぞれ遷移関数，出力関数であり，写像 $\delta: Q \times A \to Q$ および $\lambda: Q \to B$ に対応する関数である．

さて，上記の定義は一般的な多出力 Moore 形順序回路の定義であるが，これに制限を加え，装置を定義する．まず，入力記号 A のひとつの元に注目する．これは a ビットの2進ベクトルである．この2進ベクトルを2つに分割し，一方を受信部（r ビット），他方をデータ部（d ビット）とする（**Fig. 1**）．受信部の要素がすべて0のときは，データ部の値のいかんにかかわらず，入力記号の元は A_0 である．受信部の要素のいずれかが1であるときは，入力記号の元はそれぞれの量 A_i をもつ．出力記号についても入力記号と同じ制限を加える．この制限は順序回路の基本的な性質を変えることなく，独立したタイミングで動く他の順序回路との通信手段を与える．この意義はタイミングをとるための端子と，データを送受するための端子を区別することにある．

たとえば，全加算器による直列2進加算器は **Fig. 2** のような構成にすれば装置となる．入力端子 i を受信部とし，入力端子 X, Y をデータ部とする．出力端子 O を送信部，S をデータ部と考えるとよい．また，



(a) Sequential machine

(b) Unit

Fig. 1

* Description of Multi-Unit in Design Language of Computer, by Hiroshi Hagiwara (Faculty of Engineering, Kyoto University) and Yoshisuke Kurozumi (Faculty of Science, Kyoto Sangyo University)

** 京都大学工学部
*** 京都産業大学理学部

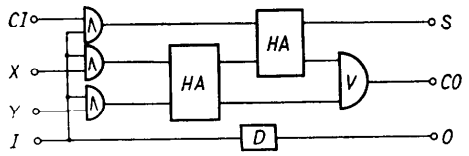


Fig. 2 Full-adder unit.

記憶装置も装置である．なぜならば，開始信号，終了信号およびデータを区別することができるからである．ところが，いくつかのフリップフロップで構成されるレジスタは装置ではない．そのままでは受信部，送信部に相当する端子がないからである．レジスタに入力ゲートと，適当な遅延子を付加すれば装置となる．

3. 装置間の通信方式の検討

前章で定義した装置間の通信について検討する．各装置は独自の処理サイクルで動作すると仮定する．また，本文の通信方式では，制御信号の方向について注目する．一般にはデータの方向に注目する場合が多いが，本文ではデータの方向はいずれでもよい．

3.1 単向通信方式

本文の単向通信方式とは，制御がある装置から他の装置への一方通行であることをいう．**Fig. 3** は装置 A から装置 B への通信を表わす．A, B 間に，通信 FF（フリップフロップ1ビット）と通信レジスタ（d ビット）を用意する．通信 FF の入出力は装置 A, B の送信部および受信部のひとつに接続する．ただし A の送信部は通信 FF をオンにし，B の送信部は通信 FF をオフにするように接続する．通信レジスタについては，A のデータ部入力を B の出力に接続するか，A のデータ部出力を B の入力に接続する．この構成で，A から B に通信する方式に2種類ある．

3.1.1 おいてけぼり方式

A が B に通信する時刻になると，A は通信 FF がオフであることを確認して，通信レジスタから入力（出力）し，通信 FF をオンにする．そのあと，A は通信レジスタを乱さない限り，直ちに次の仕事を続けることができる．一方，B は通信 FF がオンであることを知ると，通信レジスタに出力（から入力）し，通信 FF をオフにして処理に移る．B の処理は A に処理したデータを与える（A からデータを受け取り処理する）ことである．この B の状態を動作中(running)という．このような状態のとき，他の装置からの同種の通信があっても，B はその通信を保留（masked）することが多い．この通信を割込み（interruption）と呼ぶことがある．B は処理終了後，A からの次の通信待ちの状態になる．この状態が待機中（wait）である．このときは，他の装置からの通信を受け入れ（interruptable）る．この間，A はすでに B から受け取ったデータの処理を（次のデータを B に送るための処理を行なう．次のデータを入出力する前に A は通信 FF を調べる．オフならば次のデータを入出力してよい．オンならばオフになるまで，他の仕事を行なうか待機する．この待機の状態には2種類ある．

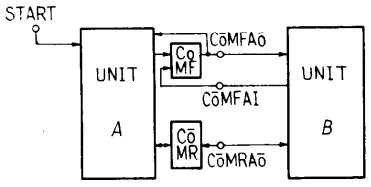


Fig. 3 Simplex communication.

① 1つは A, B のクロックパルスの長短および位相の差に原因する．たとえば，B のクロック間隔が A の10倍であれば，B の次のタイミングまで，A は最大10クロック間待つことがある．これをタイミング待ちという．

② 他の1つはが B 動作中のときである．B が動作中であれば，B の次の処理サイクルまで A は待たされる．これを動作待ちという．動作待ちはタイミング待ちに比べ非常に長いから，これを区別するため動作（busy）フリップロップを用意することがある．

この単向通信方式は直結形入出力装置の制御に使うことができる．たとえば，本体を A，タイプライタを B とすると，この方式となる．この方式による入出力装置の制御に「おいてけぼり方式」という名称がつけられている．

3.1.2 同期待ち方式

A から B にデータが伝送される場合を考える．A が B に通信する時刻になると，通信レジスタにデータを入れ，通信 FF をオンにする（A が通信可能なときには必ず通信 FF はオフになっている）．そのあと A は通信 FF がオフになるまで待機する．この待機中は同種の割込みは保留される場合が多い．B の動作については **3.1.1** と同じで，通信 FF がオンであることを知ると，通信レジスタからデータを受け取り，通信 FF をオフにして処理に移る．B によって通信 FF がオフにされたことを A が知ると，A は次の処理に移る．この方式は A の通信要求が B に受け入れられる

時刻に同期して，A は次の処理を，B は受け入れたデータの処理を始めるため，同期待ち方式という名称を与える．

3.1.1 と **3.1.2** の違いは次の点にある．前者では A が通信要求を出すと直ちに A の次の処理に移ってよいが，後者では B が受け入れるまで A が待機することにある．したがって，後者は前者よりもタイミング待ち時間だけ遅れていくことになる．2つの装置の並列処理という点では前者がすぐれている．なお，後者の方式で，B が動作終了したのち通信 FF をオフにするならば，A は B が終了するまで待つことになる．これはもっとも初歩的な制御方式であり並列処理にはならない．

一般の計算機システムでは，A が中央処理装置であり，B が入出力装置となる．A, B 間の速度比は 10^3～10^4：1 となり，並列動作を行なうと“A 待ち”になる場合が多い．そこで，A は待ち時間を他の仕事に向けるが，一方通行のため，ときどき B を監視しなければならないから，A の処理能率は向上しない．B から A への報告があればこの欠点はなくなる．これは次の二重通信方式で解決する．

3.2　半二重通信方式

半二重通信方式は2つの装置相互から情報を送ることができるが，同時には一方向の通信しか行なえない．**Fig. 4** は装置 A, B 間の通信を表わす．**3.1** と同じく，通信 FF（1ビット）と通信レジスタ（d ビット）を必要とする．A の送信部の2つを通信 FF のそれぞれセット，リセット端子に接続する．通信 FF の出力は A の受信部の1つに接続する．A のデータ部は入出力をそれぞれ通信レジスタの出入力に接続する．B についても A と全く同じ形式で接続する．したがって，A, B は通信 FF とレジスタに関して全く対称となる．

この構成で A, B が相互に通信する場合，**2.1.2** の方式を使う．すなわち，A が B に通信する場合は，A が通信レジスタにデータを入れ，通信 FF をオンにして，オフになるまで待機する．一方，B は通信 FF がオンになると，データを受け取り，通信 FF をオフにしてデータの処理に移る．B から A への通信は上の逆に行なえばよい．この方式では，通信要求が受け入れられるまで送信側が待機することを原則とする．このように A, B 間の2方向の通信ができるが，同時には1方向の通信しか行なえないから，半二重通信方式と呼ぶことができる．

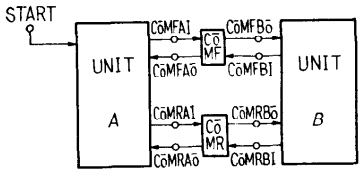


Fig. 4　Half-duplex communication

この方式では両方向から同時に通信要求を出す場合問題が生じる．すなわち，A, B 共に通信 FF をオンにして，どちらもオフになるまで待機するから，いつまでも待ち続けることになる．これがデッドロックである．この解決には，一定時間経過すると通信 FF をオフにして互いに通信を再開する方法，受付けを優先制御回路を通して行なう方法などがある．したがって，半二重通信方式は $A \rightarrow B, B \rightarrow A, A \rightarrow B$ のように通信方向と通信順序が決っている場合に採用される．

なお，単向通信方式の **3.1.1** の方式を半二重通信方式に拡張することはできない．なぜならば，**3.1.1** の方式は通信要求を出したあと，直ちに割込み可能になるから，割込みが生じたとき，自分の通信要求が相手のものかの区別がつかないからである．これを解決するために，通信方向を指示するフリップフロップを追加することが考えられるが，これよりも次の全二重通信方式の方がすぐれている．また，装置にアドレスを用意し，送信アドレスと受信アドレスを送る方法は交換通信方式に発展する．

3.3　全二重通信方式

全二重通信方式は2つの装置相互から同時に両方向へ通信ができる．単向通信方式を両方向に設備したものとみることができる．その構成は **Fig. 5** で表わされる．通信する方式は両者ともに **3.1.1** のおいてけぼり方式で行なう．この方式はもっとも広く使われており，チャネル形入出力装置の制御に採用されている．問題は同時に通信要求を出す場合であるが，さきに割込み可能になった装置が通信要求を受け入れることになる．通信レジスタも両方向に用意することができるから，完全に同時に通信することができるが，この場合，装置の受信部と送信部が並列動作をしなければな

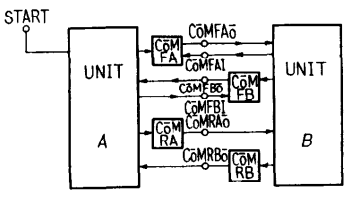


Fig. 5　Full-duplex communication.

らないため，制御が複雑になる．したがって，この方式の完全な同時通信の実用例は少ない．

この全二重通信方式の構成で，2つの装置が両方ともに **3.1.2** の同期待ち方式で通信すると，同時通信時にデッドロックになることがある．

また，一方が **3.1.1** のおいてけぼり方式を使い，他方が同期待ち方式を使うならば，同時通信時においてけぼり方式を使った装置が早く応答することになる．これにより，同時通信の優先度を決めることができる．

なお，A から B への通信要求の受入れ返答として，A の通信 FF をオフにしないで，A の通信 FF をオンにすることによって，知らせる方法が考えられる．しかし返答としての割込みか，新たな通信要求としての割込みかを区別しなければならない．さもなければ，互いに返答のサイクルを繰り返すことになる．これは通信レジスタの一部を使うことにより解決するが，処理が複雑になる．ところで，通信要求の返答については，次の通信時に通信が受け入れられたか否かを知るために必要となるもので，いつ受け入れられたかという返答時刻は重要ではない．このことは返答を割込みにする必要がないことを示す．したがって，本文のように通信 FF をオフにすることにより，返答を表わすことが望ましい．

3.4 交換通信方式

交換通信方式は多数の装置が互いに通信する場合に使われる．たとえば，**2.2** の半二重通信方式における通信と通信 FF レジスタを多数の装置から入出力できるように接続する(**Fig. 6**)．この構成ではおいてけぼり方式，同期待ち方式いずれでも可能である．しかし，どの装置からどの装置への通信であるかを指示するために，送信アドレスと受信アドレスを通信レジスタ部に含ませる必要がある．おいてけぼり方式では，自分の通信要求により割込みが生じるが，これは受信アドレスを知れば解決する．この方式では，共通の通信 FF と通信レジスタを中心として，多数の装置が互いに通信できる．しかし，同時に通信要求がある場合には問題が生じる．この解決の手段として，通信 FF に優先制御回路を付加することが考えられるが，これは交換装置とみなすべきである．中央に交換装置があれば，各装置は交換装置と通信することになり，**3.3** の全二重通信方式に帰着する．

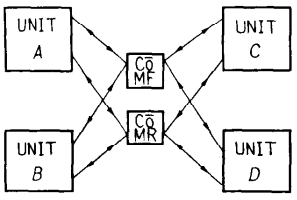


Fig. 6 Switching communication.

4. T-言語による装置の記述

前章で2つの装置 A, B 間の通信方式の検討を行なった．これらの方式を組み合わせることにより，いろいろな通信方式の構成が可能であると思われる．

さて，筆者らの計算機設計言語 (T-言語) は，計算機を言語により記述し，計算機により，設計，シミュレーションを行なうために開発された．したがって，T-言語は上記の装置間の通信方式なども十分に記述できなければならない．この章では，前章の各方式をT-言語で記述し，簡単な説明を加える．

T-言語では，装置はプロセデュアと呼ばれるプログラム単位で記述される．プロセデュアはブロック構造をとることができるから，いくつかの装置をまとめて1つの装置とみなすことにより，複数の装置が記述される．つまり，複数装置の記述はブロック構造の採用により，容易に解決できるのであるが，宣言の及ぶ範囲が問題となる．

ALGOLで使われている方法は——あるブロックで宣言された変数は，そのブロックではローカルであるが，そのブロックより内部のブロックではグローバルである．この方式は T-言語では，次の理由で採用していない．

① T-言語の宣言にはレジスタ，加算器，端子など約 10 種類がある．これらのうち，装置間の通信手段に使うものは，実用上では，レジスタと端子であるから，すべての宣言を内部ブロックに対し，グローバルに扱う必要はない．

② 多段に内部のブロックにまで宣言を及ぼす通信方式は，実用上では，ほとんどない．また，多段に扱うようにするためには，変換プログラムやシミュレーションプログラムなどを複雑にする．

③ 変換やシミュレーションは内部のブロックから行なうことが多い．したがって，内部のブロックのみを抜き出しても完全なプログラムであることが望ましい．ALGOL 形式の外から内へ主権が及ぶ方法では，内部のブロックのみ抜き出すと，未定義の変数が残る．

そこで，T-言語では次のようにした．一般の宣言はそのブロックのみに通用するものとし，内部・外部には影響を及ぼさない．特殊な宣言として，externalという外部端子の宣言子を用意し，外部のブロック(一段のみ)に対して，内部端子を宣言するものとした．グローバルに取り扱うことができるのは端子のみであるが，関数文により，他のモジュール(レジスタ，加算器など)に接続することができる．この方式により，現実の装置の外部端子との対応づけが容易になるばかりでなく，記述，変換，シミュレーションにも適すると思われる．

さて，**Fig. 7** は単向通信方式を記述した例である．

```
SIMPLEX: procedure /* SIMPLEX COMMUNICATION */
         external START;
  UNITA: procedure /* UNIT-A PROCEDURE DECLARATION */
         register COMF, COMR (1 : 8), DATA (1 : 8);
         ··· /* OTHER MODULE DECLARATION OF UNIT-A */
         external COMFAI, COMFAO, COMRAO (1 : 8), STAT;
         function COMFAO=COMF, reset COMF=COMFAI, COMRAO=COMR;
UNITAO: if STAT then wait;
         ··· /* EXECUTION STEPS */
 LINE 1: if COMF then wait;
 LINE 2: COMR=DATA; COMF=1; /* TRAP STEP */
         ··· /* EXECUTION STEPS */
         end /* UNIT-A END */
  UNITB: procedure /* UNIT-B PROCEDURE DECLARATION */
         register DATA (1 : 8);
         ··· /* OTHER MODULE DECLARATION OF UNIT-B */
         external COMFAI, COMFAO, COMRAO (1 : 8);
UNITBO: if ~ COMFAO then wait; /* INTERRUPTION STEP */
         DATA=COMRAO; COMFAI=1;
         ··· /* EXECUTION STEPS */
         end /* UNIT-B END */
         function STAT=START,
         end /* PROGRAM END */
```

Fig. 7 Program of simplex communication in T-language.

```
HALF-DUPLEX: procedure /* HALF-D'JPLEX COMMUNICATION */
         register COMF, COMR (1 : 8);
         external START;
   UNITA: procedure /* UNIT-A PROCEDURE DECLARATION */
         register DATA (1 : 8);
         ··· /* OTHER MODULE DECLARATION OF UNIT-A */
         external STAT, COMFAI, COMFAO,
         COMRAI (1 : 8), COMRAO (1 : 8);
UNITAO: if ~ STAT then wait;
         ··· /* EXECUTION STEPS */
         if ~ COMFAO then wait; /* INTERRUPTION STEP */ DATA=COMRAO; COMFAI=0;
         ··· /* EXECUTION STEPS */
         COMRAI=DATA; COMFAI=1; /* TRAP STEP */
         if COMFAO then wait;
         ··· /* EXECUTION STEPS */
         end
   UNITB: procedure /* UNIT-B PROCEDURE DECLARATION */ register DATA (1 : 8);
         ··· /* OTHER MODULE DECLARATION OF UNIT-B */
         external COMFBI, COMFBO, COMRBI (1 : 8), COMRBO (1 : 8);
UNITBO: if ~ COMFBO then wait; INTERRUPTION STEP */
         DATA=COMFBO; COMFBI=0;
         ··· /* EXECUTION STEPS */
         COMRBI=DATA; COMFBI=1; /* TRAP STEP */
         if COMFBO then wait;
         ··· /* EXECUTION STEPS */
         end
         function STAT=START, COMF=COMFAI, COMF=COMFBI, COMFAO=COMF, COMFBO=COMF,
         COMR=COMRAI, COMR=COMRBI, COMRAO=COMR, COMRBO=COMR;
         end
```

Fig. 8 Program of Half-duplex communication.

```
FULL DUPLEX: procedure /* FULL-DUPLEX COMMUNICATION */
             external START;
      UNITA: procedure /* UNIT-A PROCEDURE DECLARATION */
             register DATA (1:8), COMF, COMR (1:8);
             ... /* OTHER MODULE DECLARATION OF UNIT-A */
             external STAT, COMFAI, COMFAO, COMFBI, COMFBO, COMFAO (1:8), COMFBO (1:8);
             function COMFAO=COMF, reset COMF=COMFAI, COMRAO=COMR;
    UNITAO: if STAT then wait;
             ... /* EXECUTION STEPS */
             if COMF then wait;
             COMR=DATA; COMF=1; /* TRAP STEP */
             ... /* EXECUTION STEPS */
             if ~ COMFBO then wait; /* INTERRUPTION STEP */
             DATA=COMRBO; COMFBI=1;
             ... /* EXECUTION STEPS */
             end
      UNITB: procedure /* UNIT-B PROCEDURE DECLARATION */
             register DATA (1:8), COMF, COMR (1:8);
             ... /* OTHER MODURE DECLARATION OF UNIT-B */
             external COMFAI, COMFAO, COMFBI, COMFBO, COMRAO (1:8), COMRBO (1:8);
             function COMFBO=COMF, reset COMF=COMFBI, COMRBO=COMR;
    UNITBO: if COMFAO then wait; /* INTERRUPTION STEP */
             DATA=COMRAO; COMFAI=1;
             ... /* EXECUTION STEPS */
             if COMF then wait;
             COMR=DATA; COMF=1; /* TRAP STEP */
             ... /* EXECUTION STEPS */
             end
             function STAT=START;
             end
```

Fig. 9 Program of Full-duplex communication.

COMFAI, COMFAO, COMRAO (1 : 8) はプロセデュア UNITA UNITB にとっては外部端子であり，UNITA, UNITB について同名であるから，接続されていると解釈する．これらの端子はプロセデュア SIMPLEX にとっては内部端子である．なお，UNITA と UNITB がそれぞれ別名の外部端子を宣言し SIMPLEX で関数文により接続しても同じである．

Fig. 8 は半二重通信方式を記述した例である．親ブロックである HALFDUPLEX はレジスタ，COMF, COMR (1 : 8) をもち，兄弟ブロック UNITA と UNITB のそれぞれの外部端子と関数文により接続されている．したがって，UNITA または UNITB のみを抜き出しても完全なプログラムとなる．

Fig. 9 は全二重通信方式を記述した例である．

5. むすび

順序回路を装置として定義し，装置間の通信方式を記述するためのモデルとして，通信方式を分類した．本文の目標は，筆者らの提案した計算機設計言語によって，装置間の通信がどのように記述され，変換されるかを知ることであった．本文のモデルから出発して，現在採用されている入出力制御装置の記述を行なったが，十分に記述できることが確かめられた．

参考文献

1) 萩原，黒住：計算機設計言語，情報処理，pp. 93～102, Feb. 1971.

(昭和 45 年 10 月 30 日受付)