プログラミングー言語・基礎・実践ー　5－12
(1991. 11. 22)

ハード制御.通信に用いる埋込システムの設計手法

河野善弥

埼玉大学　工学部　情報工学科

あらまし

本報告は、ハードの制御や通信に用いるシーケンス性の強いEmbedded systemの設計手順を纏めた物で、
＊設計手順を記録に留め参考に供する事　＊良い設計手順の実体を示す事
の二つを狙いとする。　これらのシステムではシステム／ソフト／ハードと協調しながら開発が進む。システム／ソフト／ハード共に設計の本質は同一である。　良い設計の共通の鍵を説明した後、システムの設計手順／ソフトとハードの分離／機能を果たさせる為のシーケンス性を持ったソフトの設計手順を説明する。　次いで、かようなソフトを制御する為のRuntime systemの設計手順を説明する。　小さな設計のステップ毎に手順を明らかにして、ステップ毎に記録(設計ドキュメント)を残して繰返し多面的なチェックを行うべきであるとのべている。

Design Procedures for Embedded Systems used for Hardware Control and Communications

Zenya Koono

Department of Information and Computer Sciences, Saitama University

Abstract

This paper describes design procedures for embedded systems with sequential nature and used for hardware control and communications. In developments of such systems, systems design, software design and hardware design go on interacting with each other. Key factors for good design are quite the same in these three. After showing several key factors, design procedures starting from systems design, software hardware partitioning and software design with sequentialnature. The design procedures for the runtime system is also described. Throughout the paper, it is emphasizedthat following are key factors for high quality design:

Clarify design procedures for each small design steps,
Leave design records or documents at each small design steps and
Careful checks based on them.

1．緒言

本論文は次の二点を目的としている。

1．シーケンス性の強い埋込システム(ハード制御システムや通信端末など)のシステマチックな設計方法の紹介（この分野に不慣れな方々を対象として）

2．「良い設計法とは如何なる物か?」の議論の種を蒔く。（ソフト管理者やソフト研究者の方々を対象として）

本論文で、埋込システム（Embedded system)とはある効用/機能を果たしている手段としてのソフトが外部からは見えないが、内部はソフト/ハードによりシステムを構成している物をいう事とする。　この種の製品のソフト開発は純粋なソフトのみの開発より難しく、次の様な問題がよく起こる。

*製品開発が旨くいかない
*開発スケジュールが遅延する
*ソフトの品質や生産性が良くない

街角でも、釦を押しても反応時間が大きいとか、押す度毎に反応時間が変わる自動販売機など、外からみて明らかに構造不良が判る物がある。

この種の製品の開発においては、通常の事務計算(EDP)ソフトとは異なる次の様な困難性がある。

*製品の成功には優れた製品企画、良好なシステム設計、経済的なハード設計、品質の高いソフトなど総合技術を必要とする。　良い設計の観点からは的確なS/Hトレードオフといった融通性が必要であり、一方品質/日程管理等からは明確なS/Hインタフェイス等の両者の敷居の高い事が必要である。（ソフト管理者はラストランナーとして尻拭きに廻る事でなく、全体のリーダーシップを取って欲しい。)

*一般に強いシーケンス性があり、有限状態機械(FSM,シーケンシャルマシン)として取り扱い、シーケンス性の無い通常のソフトとは異なる取り扱いが必要である。　しかし、その方法を説明したテキストは極めて少なく[1,2]、又一般的な教育も殆ど行われていない。

*並行動作する抽象機械数が極めて多い(電子交換では数万になる)場合や、小型で余裕の無い場合など、Runtime systemを自己開発する。　この為OSの基本構成技術が必要である。　この様なテキストや教育も充分でない。

*ソフトの品質はハード並みに高い事が要求される。

この様に問題は単にソフトのみに留まらないので、本論文では製品計画等の最上流段階からRuntime systemまでの領域について通して検討する。

2．良い製品を創造する鍵

現状の問題は、ソフトの教育がプログラミング/技能に閉じている事に問題があるのかと感じることが多い。

例を上げるてみると、

*欧米の設計手法のテキストには顧客、最終製品等に対する配慮が余り感じられない。　自分の主張する論理のみに立脚して首尾一貫して、説を進めている。（社会／文化のしからしめる所であろう)。　読む時は「なるほど」と思っても、実際に使おうとすると壁にぶつかる。（現実の開発/設計では多くの原理/方式を適宜使い分ける必要がある。)

*日本のテキストは欧米のように個性的でなく、実践的な多くのKnow Howの個片が掲載されているが、位置ずけが理解されていないと活きてこない。

*日本的なTotal Quality Controlは日本製品の世界的な進出に大きな貢献をした。　しかし、その活動の仕方と成果の報告のみでは、十分に理解されない。

上記の何れもが真理の一面を伝えているので、これらを抽象化して見れば共通的な良い開発／設計の鍵が見出せると思う。そこで、筆者自身の体験と研究の結果を整理してみた。

段階的に緩やかに具体化する事

N. WirthはプログラミングのレベルでStepwise Refinmentを説いた[3]。　これはプログラミングのみでなく全ての段階に適用すべき事であろう。　多くの設計方法論にも説かれており、高い品質のベテランは自ら実践している。

全てを階層的に展開する事

Yourdon等は構造化設計で機能展開についての階層化を強調し[4]、又Warnierはデータ構造について強調した[5]。　インタフェイスの性質、あるいはデータとアルゴリズムの関係を考えると、両者は実は双対の関係にある事が判る。　この外にも構造化分析/ドキュメント/テスト等の階層性を強調した物が挙げられる。　身近な例ではScreen上のアイコンの展開も階層的な物が殆どである。

この事は狭義のソフトに留まらない。　ハード製造で指導的な役割を果たしたIndustrial Engineeringも階層性を重視した。　例えば、管理に使われるPART図は階層性を前提としている。　ソフト開発の組織も日本では階層的に構成する。組織論についての軍事科学の研究結果も階層的組織を支持している。　日本のソフト開発組織は殆どが階層的であり、昔Conwayの法則で指摘された所と一致している。[4,6]

Conwayの法則
「システムの構造はそれを構築する組織の構造を反映する。」
「組織によって設計される如何なるシステムの構造もその組織と同形である。」
原田,久保訳「ソフトウエアの構造化設計法」より
(米国では必ずしも階層的組織ではない。)

かように階層性に長所が現れる所以は人間の知的処理が脳の内部構造の影響を受ける為ではないだろうか?

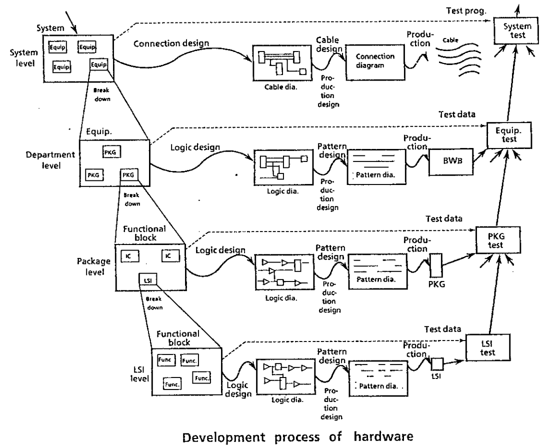
＊段階的／階層的な具体化のステップは小さい程良い[7]
品質の高い経験者は無意識で実践しているし、TQC等で品質の向上したハード/ソフト作業工程は、皆この様になっている。

＊表面的には小さなステップであるが、実質的に大きな具体化をもたらす解法が優れた解である。
通常の学校教育では結果のみで評価するが、数学ではエレガントな解が貴ばれる。　ハードの世界ではコスト低減に繋がり特許などで、無意識裏に推奨されている。　ソフトでも何ら事情は変わらない筈。

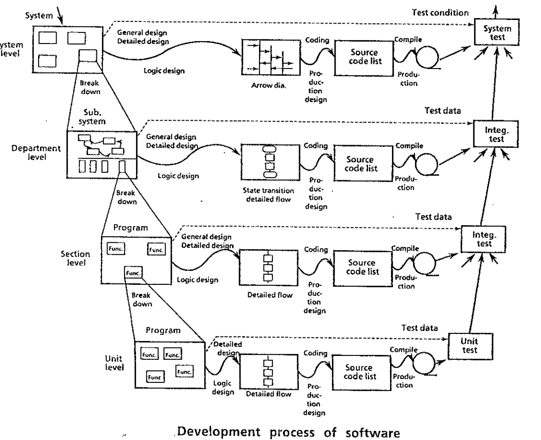
＊各具体化ステップ毎に記録を残し、レビュー／チェックを繰返し行う事[7]
これは紙の量を増やす事を意味している訳ではない。　例えば、決定にあたっては考慮した解法案を列挙して評価を明確にしておけばレビューがよく行え、プログラムに近いレベルでは処理に具体化するステップが細かく記録されていれば、チェックも容易に行え、何れも不良の事前検出率の向上をもたらす。　チェックには漏れ率があり、認知を容易にすると共に、繰返しチェックによって不良の事前検出率の向上をもたらす。

小学校で高学年に進むに従い、自然科学系の学科ではこの様に問題を解く記録を残し、チェックするように教育される。(ドキュメント不要論が説かれるのはなぜだろうか?)

＊開発の工程はハンモック状の網になる。　前半の設計文書はこの網の相互関係の中で、正しい事が必要である。
設計で文書をノードにとり、後半のテストで統合をノードにとると、図２．１の様にハードもソフトも同一様式で現される。　[8,9,10]　(相互比較するとハードの製造はソフトではコンパイル/アッセンブルあるいはコピーに相当し、ハードと異なり初期から自動化された。　ソフトもハードもの開発に関しいえば、本質に差は無い。)　設計は最終目標を各種の拘束条件を満たすようにせねばならない。　しかし、各設計工程が最終目標にそって展開される保証はなく、設計の工程毎に当初の意図は次第に変形して、下位に進むと上位に掲げた最終目標と反する設計が堂々と行われる事すら起こる。　そこで、ミクロには小さな設計ステップ毎の合致が必要であり、マクロにも最上位から始まる上位の各文書との合致が必要である。　(そこで、ミクロに見れば上位に遡ったり(Bottom up)、下位に下ったり(Top down)しながら、全体としては上位から下位へと進む。)　この目的を果たすには、担当部分の上位/関連部分の教育がよく行われ、関連文書が利用できなければならない。



Development process of hardware



Development process of software

図２．１　　ハードとソフトの開発工程　［１０］

現実に上記の各条件を満たす事は容易でない。　初心者は教えられた手順を意識しながら作業するが、習熟すると問題を与えられると直ちに最終結果が見通せる。　一般に熟練すると各手順が意識から消失するのが人間の特性である。　この様な習熟状態になった後、テスト等で最終結果が出た時に、自分がどこで何を誤ったかを認識して自己矯正できる人は少ない。　その結果、能力育成上大きなバラツキがでる。　これでは困るからは、記録を残して再発防止を通じて育成強化をはかる必要がある。　しかし、判りきった事はやりたくないという人間の特性に逆らって、記録を充分に残して繰返しチェックする事により、設計の一致性を保証するには、CASEツール等の支援が必要である。

＊管理者は設計不良に関連して修正する場合以外、段階的な当然の具体化以外の変更を作業途中で入れてはならない。
論理ハードもソフトも極度に正確でちみつな作業を必要とする。　システム/ソフト/ハードとミクロには若干位相がずれ

るが、マクロには同期しながら段階的に作業が流れ、概略から次第に詳細へと決定していく。　一束の要求/契約仕様書で一時に一切の条件が尽くされる事は実際上ありえない。

最終目的に向うこの流れを乱すと大きな混乱を呼び又モラールの低下を招く。　特にさみだれ変更は悪質である。　実際上避けられない中間の変更はストックしておき、例えばテスト/統合が終わり一応の統一性が確保できた時に一括折込む等のスケジュール上のマージンを計画しておく事が管理者の重要な役目であり、予想される変更に柔軟に対処できる構造を作り込んでおく事が技術リーダの役目である。

上記を通じて良い設計の鍵として、次の事が抽象化できる。
- ＊設計者に好ましい環境条件を作り出す事
- ＊正しい設計を行わせ、厳密なレビュー／チェックを行い指導する事
- ＊上記と技術継承の為、文書化が重要である。

他の重要な原理は、有限状態機械（ＦＳＭあるいはシーケンシャルマシン）をモデルとして設計する事などがあるが、これは次に述べる。

３．主要な設計方式原理

対象とするシステムの開発には純粋なソフトのみの開発に比べ、より多くの要素が絡む。　実際の設計では、経営段階から始まりシステム／ソフト／ハードと幾つもの大きな流れが、相互に影響しあいながら流れる。　以下に開発の為の主要な原理を説明する。

３．１目標の階層性に基ずく展開

これは軍事科学で発見された人間の基本的な行動に関する基本則で、次の様なものである。

＊ある目標Ａを設定する。　この目標Aは達成する為の幾つかの手段a1，a2，a3...に分解できる。　次にこの手段の何れかaiを新たな目標とすると、これを達成する為の手段ai1，ai2，ai3....に分解できる。　この展開は繰返し行う事が出来て下位になる程より具体的になる。全体として、最終目標を達成する為の階層的な目標/実現手段の体系(目標樹)が出来上がる。

＊上位程影響力の大きな決定を事前にするのでより高度な洞察能力が必要であり、執行の為により大きな権力が必要である。　下位は詳細で具体的な作業能力が必要である。　そこで階層的組織に対応した階層的体制がいる。

＊組織は機能別階層組織がよい。　これは分業／専門化による能力向上と、大群化による効率化を達成する。

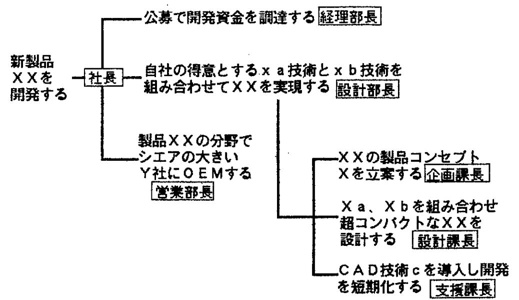


図3.1　目標の階層的展開例

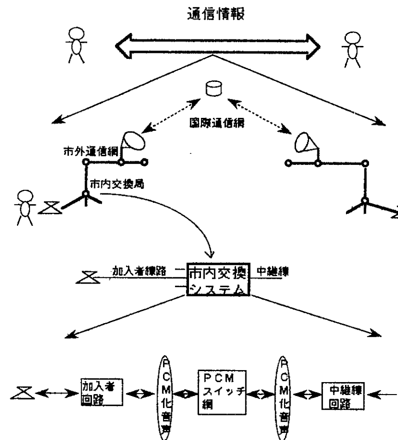


図3.2　通信情報流の階層的展開例

図３．１はその例で、上位決定は戦略、下位決定は戦術に当たる。　この原理は管理/計画/上位設計に使える。 [8]

３．２情報/事物の流れの階層的展開

次に上記で定めた具体化目標を更に具体化する。　通信関係では通信情報の流れを捕らえる。 図3.2はその例を示す。 [13]（通信情報の流通する通信網は既に階層的に構成されている。） 流れを展開していくと、この流れの中で、幾つかのノードが見えてくる。 それを更に階層的に展開する。
これは通信の例であるが、Computer Integrated Manufacturing (CIM)等では、生産の為の物と伝票の流れを採ると丁度同じ事が出来る。　対象によりある程度は変わるが一般に事物/情報の流れになる。

情報や事物の流れの上位は既に分断され名前も決まっている事が多い。　しかし細分化していくと、自分で分割/命名せねばならない。　この為にはMyersのS-T-S法を拡張して使うと良い。 [11,12]

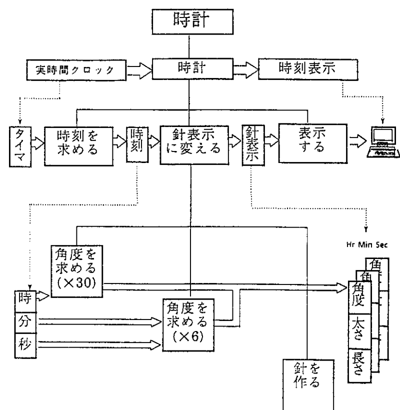


図3．3　　時計

Myersはソフトウエアは情報の変換の連鎖であり、最大抽象点に着目する事により、3部分に分断できる事を見出した。

* ソフトの中の主たるデータの流れに着目する。
* 入力側に近い部分で、入力の流れが最も抽象化される点がある。　これを最大入力抽象点と呼ぶ。
  (最も概念的にくびれた点/ノードともいえる。)
* 出力側に近い部分で、出力の流れが最も抽象化される点がある。　これを最大出力抽象点と呼ぶ。
  (別の表現では、これ以上入力に近ずくと出力の特性を失う点である。)
* 上記により、データフローは3部分に分割される。
  入力側部分は(OSのある時)制御を渡すとデータを拾ってくるので、Source、出力側部分はデータをつけて制御を渡すとデータが消失するからSink、中央の部分は変換のみであるので、Transformと名付けた。
* この分割は階層的に繰返し行え、展開をつずける程具体的になり、最後は計算処理の論理が見えてくる。

図3．3は一例を示す。[13]　この性質はソフトのデータの流れのみでなく、一般の情報などの流れに適用できる。

この様な分割を奇麗に行えば、プログラムは独りでに見えてくる。　良いアルゴリズムを展開に使えばより早くに見える。しかし、分断点の名称が実体を明確に捕らず不適切であると、段階的に展開できなくなったり、何時の間にか、当初の意図と食い違ったりする。　この分断と表現を正しく行い、機能とインタフェイスの階層的展開を上手に行うには高い抽象化能力と言語表現能力を要し、(例えば機能はXXをYYすると表現する)訓練なしで原理を説明しただけで上手に出来る人は約10%程度である。　これが上手に出来ないと構造化分析/構造化設計などは出来ない。　これは、構造化分析/設計やCASEツールを実用する上での困難性の一つである。

3．3システム/ソフト/ハードの機能分担

これまで説明した方法で、ソフトかハードかを意識せずに極力小さな具体化ステップで段階的に階層化してくれば、機能箱は単純な機能を表現し、インタフェイスは明解な切り口になる。　この様な展開の結果は機能の標準化に極めて都合が良い。

システムの対象とする/主たる事物/情報の流れを認識してそれを分割し、段階的/階層的に展開する内に、夫れ夫れの機能をソフトやハードで実現する事が認知される段階、細分化された事物や情報の流れを実現する手段として、ハードあるいはソフトが表れる段階に至る。

一般論として、ハードは外界とのインタフェイスや高速性が必要な場合に用い、ソフトは機能が大きく、それほど高速性を求められない場合に適している。　ソフト／ハードトレードオフとは、かく階層的に分割された各ブロックについて、ソフト／ハードを割り当てる幾つかの候補案の中から、最適な一案を選択する事になる。　このようにすると、通常の様に経験的に中間でソフト/ハードの切り口を設定して行う通常の場合に比べてより容易にソフトかハードかがシステマチックに決められる。

3．4ソフトやハードの中での階層的な展開

純粋にソフトやハードになった後もデータフローによる階層的な展開が行える。　これらについては周知の事なので、以下の議論は省略する。

3．5シーケンス性をもつ抽象機械の分割

対象とするシステムはシーケンス性を持つので、いろんなレベルでそれが出現してくる。　上中位での抽象的な機能箱について幾つかの状態を持つ事が意識される場合がある。この場合、状態図を書くと機能が明確に整理できる。　図3．4はその例で、データフローを分割した複数の機能箱が夫れ夫れシーケンス性を持つている。

シーケンス性がある時は、シーケンシャルマシン/有限状態機械(Finite State Machine, FSM)をモデルとして構成する。シーケンス性のある物を、複数のFSMで構成する時(Extended Finite State Machine, EFSM)と、単一のFSMで構成する場合とを比べると、前者の場合の方が論理的により単純になる。この原理は論理ゲートの価格の高かった昔は論理設計者にとって有力なゲート数低減の手段であった。　ソフトの場合にもソフト規模の減少の為の有力な手段である。[14,15,16]

大幅に低減するには、相互独立性を持つように分割する。これを最も容易に実現するには、前記のようにデータフロー上で分割するのが良い。　この場合、分割した何れの部分を取

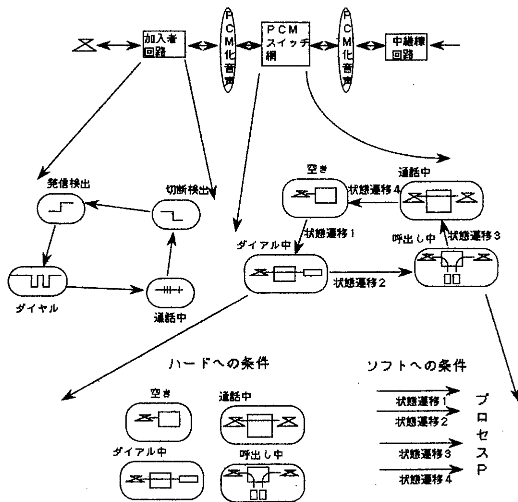


図3.4 FSMの階層的展開例

っても元の機能は構成できないから、相互独立性が強い分割になっている。

3．6状態遷移による分割

ソフト／ハードの接点になる部分では、図3．4のようにハードが幾つかの状態を持ち、これらの状態間の遷移の実現手段がソフトになる場合が多い。　この様な場合、状態はハード、状態遷移はソフトと明瞭に区分して両者を示す事により有意味なドキュメントが得られる。　以後、ソフトとハードは常に組にして管理する。　ソフト/ハード何れかのみで物事を考えてはならない。

この様に分割してもまだソフト量/規模が大きい場合がある。（例えば電子式電話交換システムなど）　この場合に更に分割する策は次のとおりである。

* 上位のデータフローに遡って更に分割する。
* 抽象的なFSMを考えて分割する。　これは抽象的であり一般に難しい。　容易化するには、目標の階層性を用い、その目的を果たす為の実現手段の群に分割する。　例えば、電話での接続サービスを
  1．接続相手を指定する
  2．指定した相手を呼び出す
  3．通話する

  などの様に分ける。

このように分割したEFSMの結合方式は

* 相互に通信させる
* Super stateを持つFSMを上位におく
* 状態と状態遷移を含む大きな部品(マクロ)とする[17,18]

などがある。

システムの立場からは、外部的なサービス機能単位に分割できると大変に好都合である。　しかし、これは一般に旨くいかない。　分割/展開を既におこなっているから、外部的なサービスの一部しか分担しておらず外部的なサービスと1対1対応には原理的にならない。

3．6FSM群の構成

分解してえたFSMはある条件で状態遷移を始める、つまり外界と通信しながら動く。　これら通信を、メッセージ(入力に着目してイベント、出力に着目してコマンド)と呼ぶ。　システムとして、複数のFSMが相互に通信しながら全体として単一の機械のように動く。　このメッセージ/イベント/コマンドの流れもデータフローの一種で、インタフェイスの階層性が必要である。　ここでもMyersの分割法が使える。　以下各FSMはプロセスと呼ぶ。　図3．5は階層的な複合系を示す。

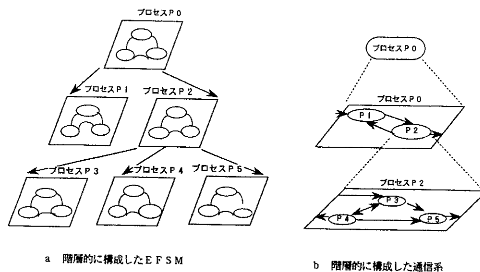


図3.5 階層的に構成した複合シーケンシャルマシン

此れ迄説明した段階的/階層的構成諸条件を守つて細分化していれば、各レベルで閉じてプロセス相互通信の全体図が書ける。その何れかのプロセスを展開すると、再び幾つかのプロセスが相互通信する様子が表れ、以下この繰返しになる。　階層性を守らないと、自分の直属上/下位でないプロセスへの通信等が生じ、面倒な問題が色々起こる。

複数のFSM間の通信ルートとプロセスは表示が容易であるが、個々のメッセージ(イベント、コマンド)の明示も必要である。この為には関係するプロセス間の信号シーケンス図が有用で、図3．6の様に段階的に具体化もできる。

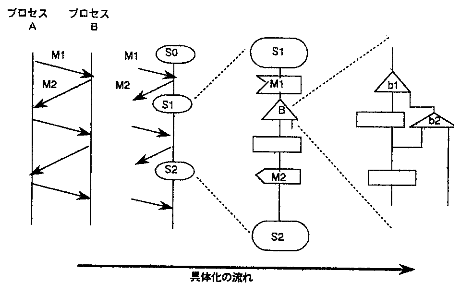


図3.6　信号シーケンス図の階層的展開例

３．８状態遷移内の分割

多くの場合、ハードの安定状態等を状態に取るので、その為の状態遷移がソフトが果たすべき役割になる。　そこで、プロセスはそれを構成するプログラムに分解される。　小さなステップで段階的/階層的に、機能/インタフェイスの階層性を持つように分解する時、最終的には単一機能/概念の小さなFSMとなり、このような場合の状態遷移からはシーケンス性は消失する。

状態遷移図上では制御の流れを示す事が必要でフロ-チャ-トが適している。　十分に小さくなる様に展開した時、フローには単純な機能箱を幾つか並べれば十分な場合が多い。
このような場合、あるプロセスで必要な機能箱の種類は限られた物になり、プログラムの為の部品群になる。
部品の中には外見は単純であるが、内部には複雑な内容を持つ場合がある。　この場合には、部品担当側で図３．６右の様に段階的／階層的な展開を継続して行う。　制御の流れに関する分岐の詳細化などは、単純なフローチャートを階層的に具体化する方が、
　＊要求条件の段階的具体化に適している
　＊人間にとり見やすい
　＊そのままプログラム化できる
　＊プログラムも奇麗になる
といった利点があり、これらは構造化風の表記は不適、GO TO 不要論は不要である。　展開した結果、一入力一出力等の条件のが満たされ内部論理が複雑な場合に限り、PAD, HCP等を利用する方が良い。

プロセスに課された機能/概念が単一で、通信のインタフェイスも段階的/階層的に十分に精選した言葉(概念)が取られていれば、部品群はそのプロセスの機能と整合した関係を持ち、単機能的になる。

３．９ＣＣＩＴＴ　ＳＤＬ

CCITT Specification and Description Language (SDL) は交換システムの図的仕様記述から出発して、現在では図及び言語表示の両形式をもち、単に交換システムのみでなく、一般のシ-ケンス性のあるシステムの仕様記述からプログラム開発レベル迄適用できる形に発展した物である。　[22]
以上述べた設計方式はSDLの上手な使い方とも言える。
これは、各種の通信システム、端末に使えるほか[23]エレベ-タ制御への適用なども試みられている。　現在この勧告に準拠したCASEツールも市販されている。 [24]

４．制御機構の設計手順

これまでの過程でソフトの処理論理は固まる。　次に複数のシ-ケンシャルマシン／EFSM全体を制御する構成の設計手順を述べる。

制御の必須条件は
　＊状態を定義し、複数種の入力(遷移原因)に応じて対応プログラムを走行させる
　＊複数のFSMがあり、或る種類のFSMは共用される
　＊ＦＳＭ／プロセス相互に任意の通信ができ、能力設計上必要な各種の通信バッファ量を設定する
更に、以下の条件もついてくる。
　＃実時間応答条件を充たす事
　＃入力を定期的に走査する事が必要で、この為数－－数十ｍｓ単位の定周期制御がとれる
　＃付加する入出力装置の為の
　　入出力割込み機能
　　入出力制御が公開され、利用者が入出力機能を作れる

現在提供されるＯＳは、シーケンス性の少ない応用を対象としており、上記のメカニズムを自分の内部には持っているが、一般には公開しておらず、上記の様な要求を充たす物は一般市場には存在しない。

巨大な電子交換システムでは市内交換と市外交換の両機能を持ち、数ｍｓ毎に入力を走査して、一時点に数十万の呼び（接続）を並行制御している。　[19]　この様な巨大なシステム、現在利用可能なＯＳを利用する工夫では賄えない小型システムでは、自分自身で用途に適合した制御機構を作るのが普通である。

この場合には、概略次の設計をする。
1. 論理的に設計しきた複数のプロセスに物理的なCPUを割り付ける
2. 処理に要求される応答時間、ピ-クに掛る過負荷等の要求より処理能力設計を行い、通信、実行制御等のキューの容量を決定する

3.状態遷移先決定機構を作る
4.実行レベルや定期走行プログラムへの制御受け渡し条件設定を行い、実現機構を定める
5.プログラム実行管理の論理構造を決定する
6.プログラム化する

４．１論理的なプロセス構成への物理CPUの割り付け
３章で説明した手順の設計に従うと図３．５の様な全体構成のシステムになる。　処理能力上に余裕の有る場合は全体が１ＣＰＵで制御されるが、複数ＣＰＵで制御する場合もある。この場合には、此れ迄論理的にのみ捕らえてきた複数のプロセスを、負荷をも考えて物理的ＣＰＵに割り付ける。

ＥＦＳＭの構成を階層的に行い、プロセス間の通信も階層を守る様に分割していれば、既存の階層構成に従って割り付けうる。　この様な配慮を欠いた場合や既存階層体系と異なる体系でCPUを割り付ける場合には、十分な検討を行い異常な動作順序にさせない為の事前確認を要する。

４．２各種のキューでのバッファ容量の設定
単一のＣＰＵで複数のプロセスを制御する場合、プロセス間の通信はメッセージのキューになる。　ＣＰＵを跨る場合は、ＣＰＵ間通信制御プログラムにより行われ、これは、入力と出力にキューを持つ。

何れの場合も必要な動特性の計算を行いキュ-容量を設定する。　メッセージの形式を統一しておけば、空きキューは一本化でき設定も楽であり、かつ空きキュー容量はトラヒック的に大群化効果が得られ総容量は小さくてすむ。

一見これらに手間をとられが、設計者に正確な事前設計を強いた代償として、正確な知識を身につける事が出来、負荷試験で初めて方式的な不良が判るという破局的なトラブルを事前に避けられ、少なくとも異常事態が生じた場合に速やかに問題解決が出来て有利である。　漫然と既存のＯＳなどを用いると、設計者は問題の所在すら知らず、プロジェクトの終末段階での過負荷試験等で初めて露見するトラブルシュートにてこずる。

４．３状態遷移決定の機構
状態遷移先の決定の原理的機構は図４．１上に示すとおりである。　図の様に明確な対応関係を持たせると、簡単なコンバータで一斉に各テーブルに記入でき、各種の設計不良が事前に予防できる。　実情に応じて変形して具体化する。　図の場合にはメッセージ中のイベントによりサーチする方式であるが、小さなシステムではイベントコ-ドでテ-ブルにアクセス出来、大きなシステムでは多段のテ-ブルを経由して状態遷移先を決定する。

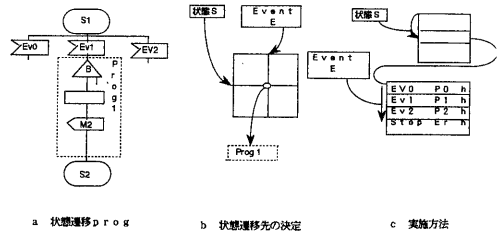


図4.1 状態遷移先の決定機構

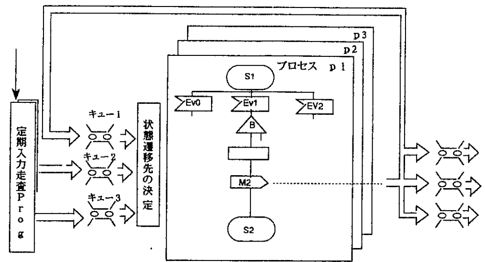


図4.2 標準化／統一化された状態遷移先の決定機構

状態遷移プログラムの制御の入口は一つであるが、出口は複数になりうる。　従って、各プログラムの出口には新しい遷移先の状態に更新する為のマクロを挿入する。

メッセージの形式に加えて、状態遷移決定の条件を標準化すると、図４．２のように状態遷移決定機構は一本化できる。

４．４実行レベルやプログラム走行条件の設定
多くの場合、状態遷移の実行はベースレベルで行い、定期走査や各種の割込みに伴うプログラム類を優先して走らせる。この為に、少なくても２以上の実行レベルが必要である。

実用状態では、過負荷入力状態が発生しうるので、予めこれに対する配慮をせねばならない。

入力プログラムの仕事量が大きい場合、重負荷状態で実行すると実時間特性を保証できなくなったりする。　そこで、作業量を平準化させる為に、対象をＮ分割して１／Ｎ倍した周期でプログラムを走行させる事などが必要になる。

ＣＰＵの使用率が高くなると、応答時間は急激に大きくなる。これが、許容されずかつ動作継続が必要なシステムでは、過負荷に近ずくと入力を制限し、入力が減ると、(ヒステリシス特性を持たせて)制限状態を解除する等の過負荷対策が必要である。　かような条件はシステム設計レベルで充分に検討して論理構造を作込み、最終段階で確定する必要がある。

運転管理指令など、通常は正規業務が優先するが、余りにも応答が遅い事は許されない場合には、監視プロセスを置き、許容時間超過が見込まれると実行順位を途中から繰り上げて、要求メッセージを再発行して高優先順位のキューにいれて、当初のメッセージを待合せているキューから取はずす等の処置を必要とする。

これらの条件により図４．３に示す実行レベルや定期的なプログラムの為の制御構造が定まる。

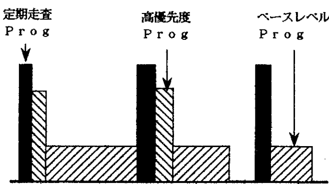


図4.3　プログラムの実行レベル

４．５プログラム実行管理の機構

最後に残された機構である。　処理のレベルに応じたキューを見て、実行する対象を定める。　図４．４はその原理を示す。　状態遷移が目的とする仕事であると考えると、遷移先の決定、実行管理はOveheadである。　FSMを多くに分割するほどこれは大きくなる。　ある電子交換システムでは、これらをFirmwareにして他社に比べて数倍大きい処理能力を達成した。　[25]

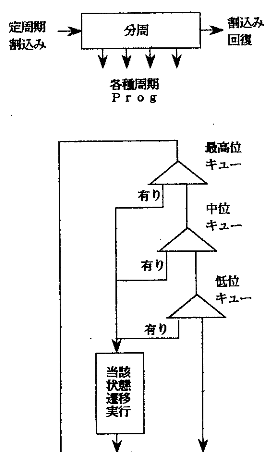


図4.4　実行管理機構

４．６その他

これらの外、高い稼働率が求められる事が多く、この為に多重化や障害対策と連続運転の仕掛けがいる。　本章に説明した事項は、いずれも本質的にはシステム設計の初めに設定され具体化が進んだ後に細部条件が確定する性質の物である。そこで、ここでは終わりの段階で説明した。

ここに記載した技術の多くは、１９６４年に個別部品を用いて最高６万回線、数百msの応答時間、４０年の長い寿命期間中の累積ダウン時間２時間以下を目標としたＮｏ．１ＥＳＳなどの電子交換システムなどの初期には幾つかの発表があった。　[19,20,21]

学会の発表は新規性を求めるからこれらの技術は学会の発表にもまた特殊分野という事でテキストにも載せられず、制御システムを専業する機関内のみに留まっているやに感じる。
ＯＳは発生的に計算機システムの有効利用を中心に発展したので、埋込システムでの利用とは異なる点が少なくない。
しかし、内部構成の基本技術は共通である。　最近はＯＳを使う面の教育のみで内部構成の教育の比重が下がっており、以外に一般の知識は低いように感ずる。

５．結び

初めに述べたように、本資料は二つの目的を持つ。

ひとつは、知識のある人には辺り前であろうが、埋込システムの設計手順を状態を持つシステムとRuntime systemの設計手順として説明した。

埋込システムはじめマイコンシステムの難しさを訴える声の中には、３章のような状態を持つシステムの設計、Runtime system, OSの基本知識を得る事により解消される物が多い。

数次の開発経験から、未経験者に最初から上記を教育しても充分には理解させられない、しかし、予めレールを引い作業させ終末段階まで前記のように段階的に順次教えると極めてよく理解させ得る。

第２の目的は「良い設計」の手法についてであった。　本文に述べた様にシステムもハードもソフトも設計の本質の差はない。　良い設計方法は段階的/階層的に小さなステップで具体化を行い、結果を記録し多角的に十分にチェックをする事により達成される。

この観点から、本報告では主要な開発のステップをシステムからプログラムに至る迄の間を通して記述した。　各ステップの間の知的な処理が創造でもあり、同時に誤りを作込む過程でもある。　小さなステップとする事は、知的な負担を減ずると共に記録チェックする事により品質を上げる鍵となる。

6．謝辞

本報告に纏めた事は引用文献以外に多くの方々との共同の研究と開発の結果である。 ご協力頂いた各位に厚くお礼申しあげる。

7．参考文献

1．Hatley, D. J. and Pirbhai, I. A. :Strategies for Real Time System Specification, Dorset House Publishing, 1987.
立田種宏訳 ：リアルタイム.システムの構造分析，日経BP社
2．野口健一郎 ：ソフトウエアの論理的設計法、共立出版社，1990.
3．Wirth, N. : Program Development by Stepwise Refinement, CACM 14-4, 221-227, 1971.
4．Yourdon, E. and Constatine, L. L. : Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design, Prentice-Hall, 1975.
原田実，久保未沙訳 ：ソフトウエアの構造化設計法，日本コンピュータ協会、1986.
5．Warnier, J. D. : Logical Construction of Programs, Van Nostrand, 1974.
6．Conway, : 1968 National Symposium on Modular Programming.
7．Koono, Z. : Build-in and Check-out of Software Errors, The Third Intenational Workshop on Software Quality Improvement, 1991.
8．Koono, Z., Ashihara, K. and Soga, M. : Structural Way of Thinking as Applied to Development, IEEE/IEICE Global Telecommunications Conference, 1987.
9．河野善弥 ：開発作業と品質向上の構造，第10回ソフトウエア信頼性シンポジウム，1989.
10．Koono, Z. and Soga, M. : Structural Way of Thinking as Applied to Quality Assurance Management, IEEE Journal on Slected Areas in Communications, Vol 8, No.2, Feb. 1990.
11．Myers, G. J. : Reliable Software through Composite Design, John-Wiley and Sons, 1976.
久保未沙，国友義久訳 ：高信頼性ソフトウエア--複合設計，近代科学社.(絶版)
12．Myers, G. J. : Composite/Structured Design, Van Nostrand Reinhold, 1978.
国友義久，伊藤武夫訳 ：ソフトウエアの複合/構造化設計，近代科学者，1979.
13．河野善弥，：構造的な思考のシステムへの適用，電子情報通信学会 研究会資料 SSE89-165.
14．桧山邦夫，水原登，草間正彦，河野善弥：分散型状態遷移方式に基ずく交換プログラムの構成法，電子通信学会論文誌，Vol. J65-B, No. 6, 785-792, 1982.
15．Hiyama, K., Mizuhara, N., Mochizuki, K. and Koono, Z. : A Software System for Electronic Switching System Using State Transition Method, IEEE International Conference on Communications 1982.
16．Koono, Z., Kondo, T., Igari, M. and Ohtsu, K. : Structural Way of Thinking as Applied to Good Design (Part 1 Software size), IEEE Global Telecommunications Conference 1991.
17．Koono, Z., Kimura, T., Iwamoto, M. and Soga, M. : A stored Program Controlled Environmental Function Tester based on FMM/SDL Design, International Switching Symposium 1987.
18．Koono, Z. : Strutural Aspect of Switching Software, 1988 Joint Conference on Communication Networks and Switching Systems.
19．高村真司，川島浩，中島汎仁，：電子交換プログラム入門，電子通信学会，1976.
英訳判 ：Software Design for Electronic Switching System, Peter Perigrilus, 1979.
20．Special Issue on No.1 ESS, Bell System Technical Journal, Sept., 1964.
21．Special Issue on No.4 ESS, Bell System Technical Journal, Sept., 1977.
22．CCITT : Recommendation Z. 100, Specification and Description Language, 1988.
23．Koono, Z., Yonezu, Y., Iwata, Y. and Hosoda, M. :Experiences in Applying SDL, Fourth SDL Forum or SDL '89 The Langauage at Work, North-Holland, 1989.
24．EIE データ社 ：SDLソフトウエア開発ツール SDT SDT.
25．Sandberg, K. and Grandberg, S. E. : New Central Processor Architecture for AXE, International Switching Symposium 1984.