設 計 自 動 化 76－9
（1995．7．20）

拡張HCPチャートを用いたハードウェア設計支援手法の提案

森田 由美子　塩見 彰睦　今井 正治

豊橋技術科学大学　情報工学系

あらまし

本稿では，プログラム設計図法 HCP チャートをハードウェア記述向けに拡張した，“拡張 HCP チャート (XHCP チャート)” を用いたハードウェア設計支援手法の提案を行なう．XHCP チャートは，ハードウェアの動作レベルの記述に適し，HDL 記述と対応がとれるので，設計期間の短縮，設計の質，検証およびデバッグの効率などの向上に効果があると考えられる．また，この XHCP チャートを用いた設計支援ツールについても述べる．

和文キーワード

拡張 HCP チャート (XHCP チャート)，HCP チャート，ハードウェア記述言語，設計支援，動作レベル記述

Proposal of Design Support Method for Hardware Using Extended HCP-chart

Yumiko MORITA　Akichika SHIOMI　Masaharu IMAI

Department of Info. and Comp. Sci.
Toyohashi University of Technology

Abstruct

This paper proposes “Extended HCP-chart(XHCP-chat)”, which is an extension of HCP-chart toward hardware design support. XHCP-chart will be effective to (1)design turn-around-time(TAT) reduction, (2)quality design, and (3) verification and debugging, of hardware, because XHCP-chart is suitable for behavioral description of hardware and corresponds to HDL description. Design support tool based on XHCP-chart is also discussed.

Key words

Extended HCP-chart(XHCP-chart), HCP-chart, Hardware Description Language, Design Support, Behavior Level Description

1 はじめに

近年ハードウェア設計の分野では，VHDL，Verilog HDL などのハードウェア記述言語 (HDL) を用いてハードウェアの機能を記述し，論理合成，レイアウト合成を行なうことで，ハードウェアの設計を行なうことが可能となった [1]．HDL によるハードウェア設計が，プログラミング言語によるソフトウェア設計に類似していることから，ソフトウェア工学の手法をハードウェア設計の分野に応用しようとする研究が行なわれている [2, 3]．しかし，ハードウェアの記述とソフトウェアの記述とで意味 (セマンティクス) が異なり，簡単にソフトウェア工学の手法を応用することができない．

また HDL 記述は，プログラミング言語によるソフトウェアの記述と同様に，全体の概念や処理間・データと処理の関係などが直観的に理解しにくい．そこで，直観的に設計ができる，図を使った設計支援ツールが提案されている [4, 5, 6]．これらのツールは，状態遷移図，ブロック図，論理回路図等を用いて，設計・検証を行ない，HDL 記述を生成する．しかし，これらはハードウェアのデータフローや構造を記述するもので，ハードウェアの動作を手続き的に記述するための図ではない．また，これらの図は，生成する HDL 記述とのセマンティック・ギャップが大きいため，HDL 記述に不具合が起こった場合，図との対応が難しいという問題もある．

本研究では，日本電信電話株式会社 (NTT) で開発されたプログラム設計図法：HCP チャート (Hierarchical and ComPact description chart)[7, 8] に着目し，これを用いて HDL によるハードウェアの設計支援をすることを考えた．

HCP チャートは，データと処理を階層的に詳細化し，データと処理の関係を表現することができる．また，プログラムコードとの対応がうまくとれるため，プログラムの記述後の検証で不具合が起こった場合もチャートに戻り，変更を行なうことが容易である．

本稿では，拡張した HCP チャートを用いたハードウェア設計手法を提案する．拡張 HCP チャートは HCP チャートの特徴を生かし，ハードウェアの記述に必要な並行処理や同期などの記述を追加したものである．これを用いて，ハードウェア設計や，デバッグの効率の向上が期待できる．また，拡張 HCP チャートを組み込んだ設計支援ツール (PAN/XHCP) についても提案する．

まず 2 節で HCP チャートについて述べ，3 節で HCP チャートを用いたハードウェア設計を述べその問題点を明らかにする．4 節では，HCP チャートの拡張方法について述べ，5 節で拡張 HCP チャートを用いた設計支援ツール (PAN/XHCP) の概要，最後に 6 節でまとめと今後の課題を述べる．

2 HCP チャート

HCP チャートは日本電信電話株式会社 (NTT) 電気通信研究所で開発されたプログラム設計図法であり，大規模なソフトウェアの設計に有効であることが知られている．

HCP チャートの記述例と用語を図 1に示す．

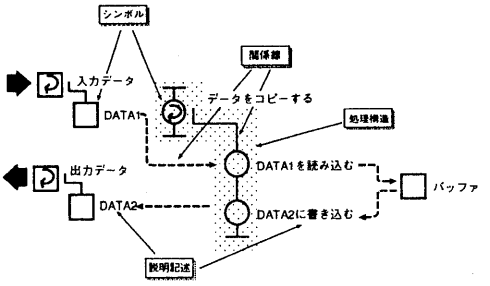

図 1: HCP チャートの例と用語

シンボルにはデータを表すデータシンボルと処理を表す**処理**シンボルがあり，それぞれの内容を自然言語を用いて**説明記述**を行なう．**関係線**には，**制御線**とデータフロー線がある．制御線は，シンボルを結びデータの構造と処理の制御を階層的に表す．データフロー線は，あるデータを扱う処理がある場合に，そのデータと処理のシンボルを結ぶ線であり，データの流れを表す．**処理構造**は，処理の始まりを表す T 字型のシンボルからはじまり制御線で結ばれた処理の連なりである．

HCP チャートの主な特徴を以下に示す．

- 処理の階層と制御の明示
- データの階層的明示
- データと処理の関係の明示
- 実際のプログラミング構造と対応

HCP チャートを使ってプログラミングを行なう場合，上記の特徴から以下のような効果が考えられる．

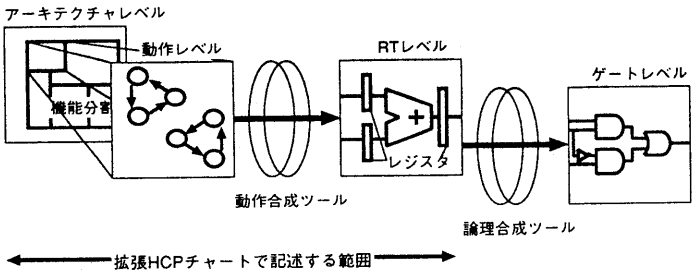

図 2: ハードウェアの設計レベルと拡張 HCP チャートで記述する範囲

(1) **プログラミング技術をサポート**
段階的詳細化により設計を進めるため，モジュール分割などが簡単にできる．

(2) **視覚的・直観的に分かりやすい図**
記号や矢印を使ったグラフィカルな図によって，処理間，処理-データ間の相互関係を表現しやすい．

(3) **概念レベルからソースレベルまでの広い設計範囲**
階層的に詳細化を進めていき，同一の設計環境上で，抽象的な概念レベルから，実際のソースプログラムとの対応がとれるレベルにまでの記述を詳細化することが可能である．

(4) **レビューの効率向上**
人に分かりやすい図を提供しているため，設計者以外の人も理解することができる．

(5) **デバッグの容易化**
HCP チャートの記述はソースプログラムとの対応がとれているため，不具合が起こった場合も，疑わしい部分を絞り込んで検査することができる．

3 HCP チャートを用いたハードウェア設計支援

本節では，HCP チャートによるハードウェア設計支援で取り扱う設計レベルと，支援の際に問題となる HDL 記述との対応について考察する．

3.1 設計レベル

ハードウェアの設計は，抽象度の高いアーキテクチャレベルの設計から実際の回路のレベルへと段階的に行なっていく．実際のハードウェアのイメージが希薄なアーキテクチャレベルと動作レベルには，明確なレベルの切り分けは難しいが，アーキテクチャレベルでは概念的な機能の分割を行ない，動作レベルでは分割された機能を詳細に決定する．次に RT レベルではレジスタだけはハードウェアに 1 対 1 対応で直接的に明確にし，レジスタ間の機能を設計する．RT レベルの HDL 記述は，論理合成ツールなどを用いて，ゲートレベルの記述を生成し，レイアウト合成を行なって，実際の回路を生成する．

HCP チャートで対象とするのは HDL を用いた設計支援の立場から，アーキテクチャレベルから RT レベルまでを取り扱う．HCP チャートで記述するハードウェアの記述レベルの範囲を図 2 に示す．

HCP チャートで，それぞれのレベルの記述をする場合を考える．

- **アーキテクチャレベル**
設計の対象とするハードウェアが「何をするものか」そのためには「どうするか」という具合に，ハードウェアの機能を分けていく．これは，HCP チャートの処理の詳細化の過程と一致し，ハードウェア設計のモジュールの分割に相当する．

- **動作レベル**
先のアーキテクチャレベルでの設計と同様に，実際のハードウェアを意識しない抽象度の高い設計であるため，HCP チャートに適したレベルである．ここでは，モジュールの動作を記述し，必要であればさらにモジュールを機能分割する．

- **RT レベル**
レジスタをデータを表すシンボルで，レジスタ間の機能を処理を表すシンボルで記述する．

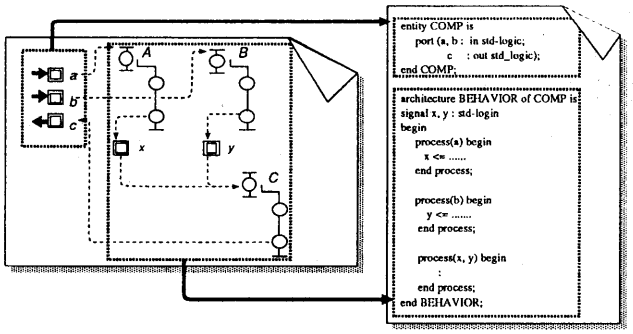

図 3: XHCP チャートと VHDL 記述との対応

RT レベルで記述するレジスタ間の機能は，構造の記述であったり，記述量の少ない動作の記述であったりする．それに対し，アーキテクチャレベルや動作レベルで記述するのは抽象度の高い逐次的な処理の記述で，よりソフトウェアのプログラミングに近い記述となり，HCP チャートの特徴を生かすことができる．このことから，HCP チャートは概念的なアーキテクチャレベルや動作レベルの設計に適していると考えられる．

また最近では，動作レベルの記述から RT レベルの記述を生成する動作合成ツールも実用化されている [9]．今後，大規模なシステム設計には動作レベルの設計が重要となり，HCP チャートによる上流設計の支援は有効であると考えられる．

3.2 HDL 記述との対応と問題点

HCP チャートはチャートの記述とソースプログラムとの対応が取れていることが知られている．

ここでは，HCP チャートを，HDL によるハードウェア設計支援に用いる場合のチャートの記述と HDL 記述の対応を代表的な HDL である VHDL[10] を用いて考察し，HCP チャートを用いる場合の問題点を明確にする．それらの問題点から HCP チャートを拡張し，ハードウェア記述向きの拡張 HCP チャート (XHCP チャート) を考える．XHCP チャートと VHDL 記述の対応を図 3に示す．

VHDL 記述では，ある 1 つの機能を遂行するハードウェアを記述する場合，その外部とのインタフェースを表すエンティティ部とその動作や構造を表すアーキテクチャ部を別々に記述する．HCP チャートにおけるインタフェースデータはエンティティ部に対応し，処理を表す部分がアーキテクチャ部に，内部データは，アーキテクチャ部で使われるデータに対応させることができる．

VHDL 記述のデータには次の 2 つがある [11].

信号 ハードウェアの結線の抽象記述で，値を保持したり，コンポーネントの起動・接続に使われる．インタフェースとなる入力・出力・入出力信号，さらに内部信号がある．

変数 動作中の計算などに用いられ，ハードウェアとの直接の対応づけはできない．

HCP チャートにはデータを表すシンボルが 1 つしかないため，信号と変数を区別することができない．

さらに，VHDL 記述のアーキテクチャ部には，並行に処理される同時処理文を記述する．しかし，HCP チャートはノイマン型の計算機による逐次的なプログラムの設計を対象としているため，並行な処理や，それに伴う同期を記述することができない．ハードウェアの設計を行なうためには，並行処理や同期の記述が必要である．

また，同時処理文中の逐次的な処理を表す PROCESS 文は，センシティビティ・リストと呼ばれる複数の信号の変化によって起動される．HCP チャートでは処理を起動させる記述についても定義されていない．

本研究では，これらの問題点を解消するために，HCP チャートをハードウェア記述向けに拡張した拡張 HCP チャート (XHCP チャート) を提案する．

XHCP チャートの拡張方法を次節で述べる．

4 XHCP チャートの拡張

拡張 HCP チャート (XHCP チャート) は，HCP チャートに信号，並行処理，同期，処理の起動の記述を追加し，ハードウェア記述向きに拡張したものである．それぞれについて，VHDL 記述での対応についても述べる．

4.1 信号

ハードウェアのデータである信号と変数を区別するために，信号を表すデータの新しいシンボル (◫) を追加する．これによって，信号と変数の区別ができる．それぞれ，データの記述方法に従い，データの階層化や繰り返しデータの記述を行なう．

4.2 並行処理

並行処理を表す記述として次の 2 つの方法を追加する．

(1) 並行処理を表すシンボルの追加

(2) 複数の処理構造の記述許可

(1) は，並行処理を表す新しいシンボル (◎) を追加する方法である．並行処理の制御の推移は図 4 のようにして表す．同時に行なっている複数の処理が，すべて終了するまで次の動作には移行せず，このシンボルの下階層のすべての処理は，処理の始まりと終りで同期する．このシンボルを使って表される並行処理は，ある機能を表す一連の処理内で，並行的な処理をする場合に用いることができる．

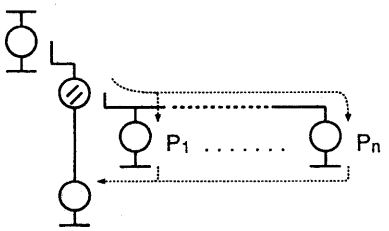

図 4: 並行処理の制御の移行

一方 (2) は，コンポーネントにおけるいくつかの機能が並行に処理する場合の記述を行なう．1 つの機能は一旦起動されると，それぞれが並行に処理することになる．このような，処理の記述のために XHCP チャートでは次のようにする．HCP チャートでは，1 つのチャートに 1 つの処理構造しか記述することができないが，XHCP チャートでは，コンポーネント内で並行に処理するいくつかの機能を 1 つのチャートに表すことが必要になる．そこで，XHCP チャートでは，コンポーネント内の複数の機能を記述できるように，1 つの機能を 1 つの処理構造として記述し，1 つのチャートに複数の処理構造を記述できるようにする．

4.3 同期

同期を記述するために wait，つまり，その時点で処理を中断し，再起動の条件が整うまで待つ処理を使う．再起動の条件に，処理や同期をとるための信号を記述しておけば，何と同期をとるかを明示できる．また，ある時間だけ処理を中断する処理の記述もこのシンボルを使って記述できる．

このため，wait を表す新しいシンボル (△) を追加する．wait を使った記述例を図 5に示す。並行に処理している処理 1 と処理 2 が，信号 a を介して同期をとっていることを示している．これらは 1 つのチャート内，つまりコンポーネント内で記述される処理間で適応できることとする．

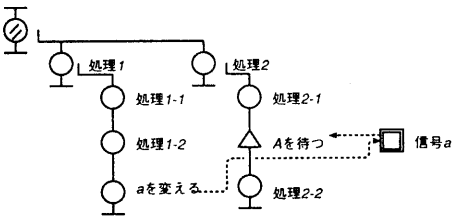

図 5: wait 処理の記述方法

この wait の使い方は次の 3 つがある．

- **信号や変数の変化を待つ場合**
  信号や変数を表すシンボルと wait を表すシンボルをデータフロー線で結ぶ．
- **処理の終了を待つ場合**
  その処理を表すシンボルと wait を表すシンボルをデータフロー線で結ぶ．
- **一定の時間を待つ場合**
  再起動の条件として時間を記述する．

HCP チャートでは，データフロー線はデータと処理を結ぶだけであったが，XHCP チャートでは，wait に向けられるデータフロー線に限り，処理を表すシンボルから引くことができることとする．

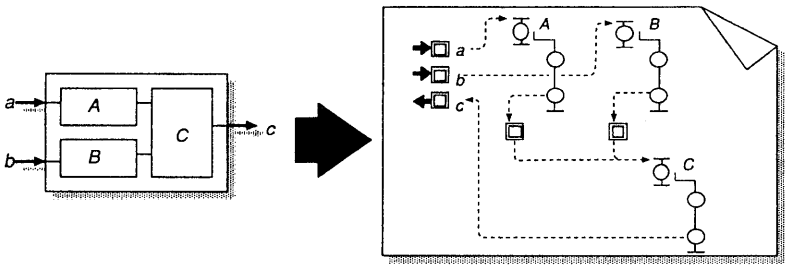

図 7: 全体の記述方法

4.4 処理の起動

処理の起動を表すために，処理の始まりを表す入口シンボルに起動する信号からインタフェース線を引く (図 6). XHCP チャートではさらに抽象的な記述に対応するために，他の処理からの起動を，処理を表すシンボルから入口シンボルにデータフロー線を引いて表す．入口シンボルにデータフロー線が引かれていない処理は，その処理を含むコンポーネントが起動されるのと同時に，その処理も起動される．

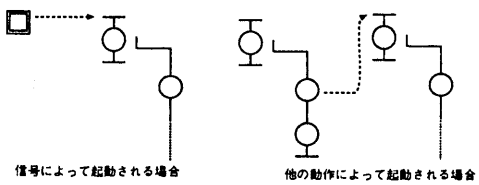

図 6: 処理の起動

4.5 ハードウェア全体

以上のような記述を用いて，ハードウェアの記述を行なう．1 つのチャートには，ハードウェアの単位であるコンポーネントを記述する．図 7に例を示す．コンポーネントの記述は，入出力を表す信号とそれらを取り扱う 1 つまたは複数の処理で構成される．コンポーネント内の処理は，階層的に詳細化され，詳細化された処理が他の動作を起動したり，同期をとることをチャートで示す．

図 7の左側に示したコンポーネントを XHCP チャートを用いて記述すると右側の図のようになる．このコンポーネントは，A，B，C の 3 つの処理と入力信号 a，b と出力信号 c を持ち，処理 A，B はそれぞれ入力信号 a，b によって起動され，それらの結果を示す 2 つ信号が処理 C を起動し，出力信号 c を得ることが分かる．

5 PAN/XHCP

ハードウェアの記述向けに拡張した HCP チャートを用いた，ハードウェアの設計支援ツール (PAN/XHCP) を実現する．

5.1 概要

PAN/XHCP の概要を図??に示す．
PAN/HCP を用いて次のように設計を行なう．

(1) **チャートの記述**　拡張 HCP チャートエディタを用いて，アーキテクチャレベルの設計で機能分割をし，その機能について動作レベルまたは，RT レベルのチャートの記述をする．

(2) **検証**　記述したチャートを次の 2 つのことについて検証する．

  (a) シンタックス・チェック
  チャートが規則どおりに記述されているかを調べる．制御線や必要なシンボルを書き忘れていないか，存在しないところを参照していないかなどを検証する．

  (b) セマンティクス・チェック
  チャートが仕様どおりに記述されているかを調べる．動作が意図したときに起動されているかなどを検証する．

(3) **HDL 記述**　詳細化した動作に対応した HDL 記述を行なう．

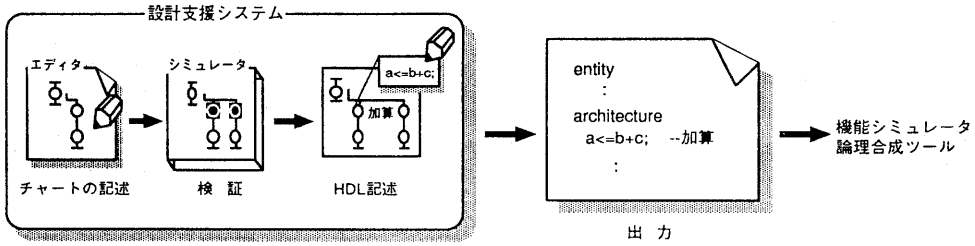

図 8: 設計支援システムの概要

(4) **出力**　インタフェース部，動作部の HDL 記述をチャートに自然言語で書かれた処理説明をコメントとして付加して，1 つの HDL 記述として出力する．

この出力の HDL 記述を各言語の機能シミュレータを用いて検証し，論理合成を行なう．

5.2 実現

XHCP チャートを用いた設計支援ツール PAN/XHCP を EWS 上に実現する．PAN/XHCP は，HCP チャートエディタ PAN/HCP をもとに，XHCP チャート向けに拡張を行ない実現する．現在，開発中の PAN/XHCP の実行画面を図 9に示す．それぞれの機能の実現方法を次に述べる．

(1) **チャート記述機能**　XHCP チャートで追加したシンボルなどが記述できるように，メニューなどを変更して実現する．

(2) **検証機能**　シンタックス・チェックを行なうチェックを，PAN/HCP に組み込まれているチェック機能を XHCP チャート向きに変更して実現する．
セマンティクス・チェックを行なうシミュレーション機能を，ペトリネットのシミュレータを応用して実現する．このシミュレーションでは，処理の移行を示すことで検証を行なう．

(3) **HDL 記述機能**　チャートのシンボルを選択し，その部分の HDL 記述を行なうエディタを立ち上げる機能を実現する．

(4) **出力機能**　PAN/HCP に組み込まれているひな型出力機能 (説明記述を C 言語のコメントとして出力する機能) を，HDL のコメントを出力できるように拡張し，HDL 記述とともに出力する．

6 おわりに

本稿では，ソフトウェアの設計に用いられる HCP チャートを，HDL を用いたハードウェアの設計に応用することを提案した．HCP チャートをハードウェア向きに拡張した XHCP チャートを用いたハードウェアの記述方法を示し，XHCP チャートが，抽象度の高い動作レベルの記述を行なうのに適し，HDL 記述との対応がとれているをことを示した．そして，XHCP チャートを用いたハードウェア設計支援ツールの概要とその実現方法について述べた．今後，EWS 上にシステムの実現をし，有効性の検証を行なっていく予定である．

拡張 HCP チャートは，ハードウェアだけでなく，複数の機能を並列に実行する大規模なシステムの設計や，ハードウェア/ソフトウェア・コデザインの分野にも有効であると考えられる．

謝辞

本研究を進めるにあたり，御助言をして下さった青山義弘助手，本間啓道先輩ならびに今井研究室の諸兄に感謝致します．なお本研究の一部は，科学研究費補助金 奨励研究 (A)07780251，財団法人 堀情報科学振興財団の研究助成による．

参考文献

[1] 安浦寛人，“論理合成時代のハードウェア記述言語”，情報処理，Vol.33，No.11，pp.1236-1243，(Nov.1992)．

[2] Kevin Covey, Sandra Murdock, Thomas Shiple, “Two-phase Logic Design by Hardware Flowcharts”, In Proc. Int. Conf. Computer Design, (1994).

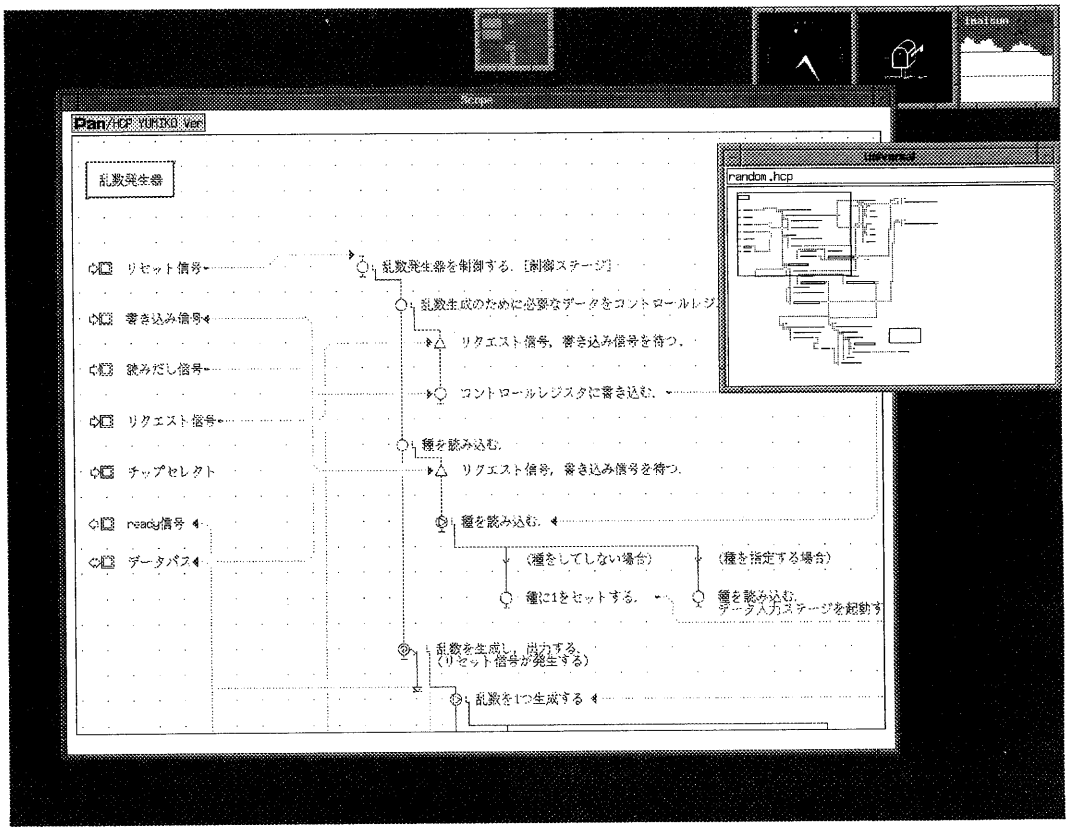

図 9: PAN/XHCP

[3] 野村雅也，岩井原瑞穂，“プログラムスライシングを用いた HDL 記述の検証” 情報処理学会研究報告 94-DA-75-2，(1994)．

[4] 中谷和恵，高井裕司，松本道弘，岩崎知恵，村岡道明，“動作機能入力システム Bchart の評価”，情報処理学会 DA シンポジウム’93，pp.97-100，(1993)．

[5] 野地保，濱田英幸，水本勝也，高坂宏之，青木洋，中村彰，“ブロック図による HDL ベース設計環境”，情報処理学会研究報告，94-DA-71，pp53-60，(1994)．

[6] 菅谷和伸，山田孝光，野田桂子，岡善治，“グラフィックとハードウェア記述言語とのミックス入力による設計手法の評価” 情報処理学会 DA シンポジウム’94，pp.27-30，(1994)．

[7] 花田 收悦，“プログラム設計図法”，企画センター，(1983)．

[8] 長野宏宣，浅見秀雄，忠海均，“HCP チャート–階層化プログラム設計図法”，電気通信協会，(1993)．

[9] “基本回路構成の設計を自動化し，最適な RTL データの作成が可能に”，日経エレクトロニクス，1994.8.1，pp58-65，(1994)．

[10] IEEE，“IEEE Standard VHDL Language Regerence Mannual”，IEEE，(1994)．

[11] Rober Lipsett 他 著，杉山尚志 監訳，“VHDL: 言語記述によるハードウェア設計へのアプローチ”，マグロウヒル，(1990)．