NEC μPD7220/72120 Related Public Documents

- ISSCC (International Solid-State Circuit Conference)
- μPD7220A User's Manual
- Nikkei Electronics Magazine (μPD7220)
- Transistor Gijutsu (Technology) Magazine (μPD7220)
- Transistor Gijutsu (Technology) Magazine (μPD7220A)
- Nikkei Electronics Magazine (μPD72120)
- μPD72120 User's Manual

Go to https://www.oguchi-rd.com/LSI%20products.php to get more detailed NEC μPD7220/72120 related information such as;

“Logic Schematics”, “Design Notes”, “Evaluation Board Schematics”, “Evaluation Software”, “Silicon Die Photos”, “Newspaper”, “Magazine”, and so forth.

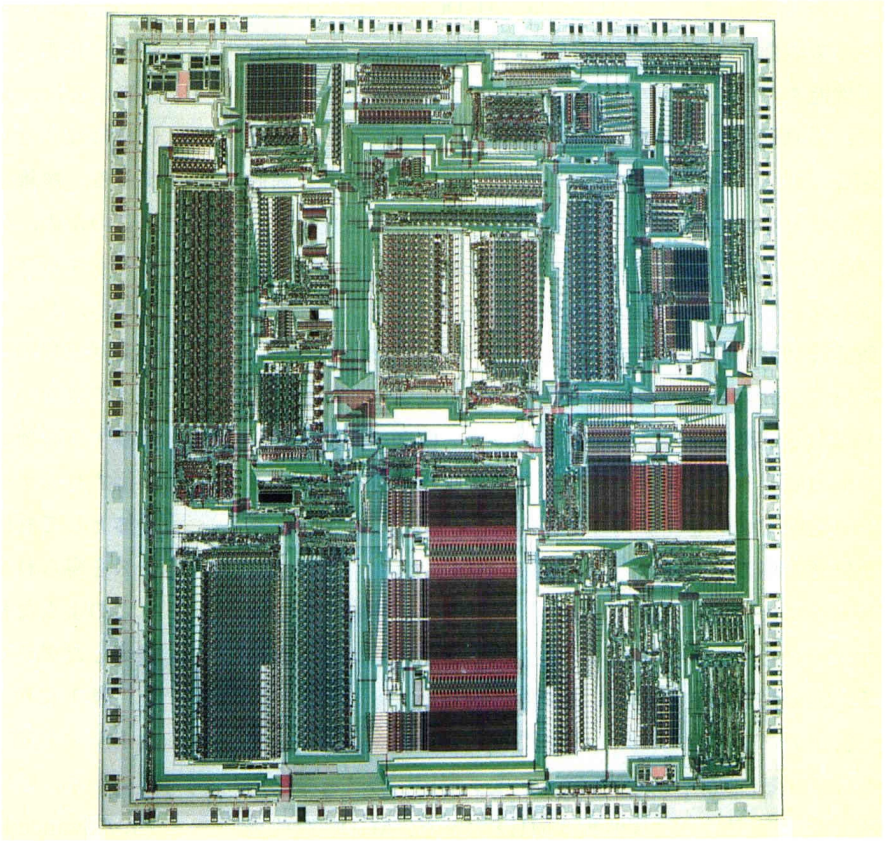
Go to https://www.oguchi-rd.com/patents.php to get patent information including NEC μPD7220/72120 related patents.

論文

μPD7220後継のグラフィックス・コントローラLSI，コピーや塗りつぶし機能を強化

小口 哲司
大内 光郎
堀口 立二
千葉 俊和
日本電気 マイクロコンピュータ技術本部
鵜野 敬史
同 第一LSI事業部

日本電気は，グラフィックス・コントローラLSIの新機種AGDC（μPD72120）を開発した。1982年に発売したGDC（μPD7220）の後継機で，ビットマップの処理機能を強化した点が特徴。拡大・縮小および回転を含んだ矩形領域のコピー機能，三角形，台形の塗りつぶし機能などを備えている。LSI内部には描画用プロセサとは別に描画の前処理を実行するプリプロセサを搭載した。CPU，プリプロセサ，描画プロセサによる3段のパイプライン処理で，システムとしての性能を上げている。表示メモリの容量は最大32Mバイトで，デュアル・ポート・メモリを使うことができる。最大8MHzで動作する。（本誌）



μPD7220後継のグラフィックス・コントローラLSI，コピーや塗りつぶし機能を強化

1981年2月に開催されたISSCCで，グラフィックス描画/表示用LSI μPD7220（GDC）を発表した。それ以来，この製品はパーソナル・コンピュータをはじめとするOA機器，魚群探知機，写植機，衛星地上端末装置など幅広い分野に応用されてきた。GDCは，システム・バスから分離した最大512Kバイトの表示メモリを直接制御できる。直線，円弧，グラフィックス文字などを高速に描画する機能を備えている[1]。GDCを用いると直線，円弧などの1次元的な描画は格段に速くなった。しかし，塗りつぶしやビットマップ・メモリに対する文字フォント展開など，いわゆる2次元的な（面に対する）描画機能は備えていない。このため，CPUでその機能を補完せねばならなかった。

このたび開発したμPD72120（AGDC）では，CPUによる機能補完を最小限に抑えることをねらった。豊富な描画機能をグラフィックス・コントローラに付加するとともに，描画速度を高速化した。図形描画機能だけをとればGDCと比べて顕著な差異はない。ただし，楕円，弧，扇形，弦形などを一つのコマンドで描画できるように拡張した。

GDCに欠けていたビットマップ処理機能を積極的に搭載した。表示メモリ間のデータ転送（コピーあるいはbitblt機能），任意閉領域内の塗りつぶし（ペイント機能），円，楕円，三角形，台形内の無条件塗りつぶし（フィル機能）などである（**図1**(a)）。このうちのコピー機能は，文字フォントの展開やウインドウの表示に重要な役割を果たす。コピー途中で図形を任意角度で傾斜，回転したり，制限はあるが任意倍率で拡大・縮小する機能も追加している（図1(b)）。ビットマップ機能の強化に伴い，主記憶上に格納した1次元データを表示メモリ上に2次元的に展開するプット機能，その逆の動作をするゲット機能も付加した。

汎用のプロセサを使用してもこれらの機能を提供することはできる。しかし，十分な描画速度を期待することは現実的でない[4]。クロック周波数8MHzのとき，AGDCの代表的な描画速度を以下に示す。直線描画は，1ドットまたは1ピクセルの描画を500 nsで実行する。コピー機能を用いれば，1秒間に1万3640字の24×24ドット構成の文字フォントを展開できる。

文字表示からグラフィックス表示へ急速に移行している

メモリの価格が下がるにつれ，ディスプレイの表示形態は文字表示からグラフィックス表示へと急速に変化してきた。ワードプロセサなどでもグラフィックス・データを自由に取り扱えることが最低条件とさえいわれ始めている。ところが，グラフィックス表示は，大きなメモリ容量を必要とし，制御も難しい。画面上の1文字を変更する場合，文字表示では文字コード・メモリの内容を1バイトまたは1ワード書き換えるだけですむ。一方，グラフィックス表示では，文字フォントを表示メモリ上に展開し直さなければならない。CPUによる処理では書き換え速度が遅すぎる。複雑な図形の塗りつぶしも，CPUには荷が重い。

このため，ゲートアレイなどを用いた専用回路を使ってビットマップに対し高速に描画する試みが成されてきた。しかし，ゲートアレイ自身の集積度や速度の制約から，たとえば三角形塗りつぶしなどのマクロな描画を実行することはできない。CPUが，細かい区切りで逐一，その描画に関与する必要があった。細かい区切りを単位に描画速度を評価すればゲートアレイは高速である。しかし，システム全体として評価した場合には，期待したほどのパフォーマンスを得られない場合が多い。

このような背景から，GDCの発表以後，描画機能などを強化したグラフィックス・コントローラLSIが多数登場した。**表1**にコントローラの代表的な機能を取り上げて，主なLSIを比較してみた。これらの製品は各社の設計思想の違いから，それぞれ異なった特徴を備えている。Am95C60（米Advanced Micro Devices, Inc.）とAGDCはハードウ

エアによる高速描画機能の実現を重視している。HD63484（日立製作所）はさらに表示制御機能を生かそうとした。82786（米 Intel Corp.）は主に表示制御機能を充実しようとし，TMS34010（米 Texas Instruments Inc.）は汎用型プロセサを内蔵し自由に描画ソフトウエアを作成できるようにしている。

グラフィックスに関してもう一つの重要な動きがある。ライン・バッファを内蔵したデュアル・ポート・メモリの出現である。最初の製品は，シリアル・メモリ型のライン・バッファを内蔵した64 Kダイナミック RAM であった[2]。その後，ライン・バッファをランダム・アクセス型とし，リアルタイム・データ転送機能，ポインタ・コントロール機能，ライト・パー・ビット機能などを盛り込んだ256 Kデュアル・ポート・メモリが発表された[3]。

デュアル・ポート・メモリを表示メモリとして使用すると，描画可能なタイミングが格段に増加する。これは，連続描画が可能なコントローラの実力を引き出すのに有効である。

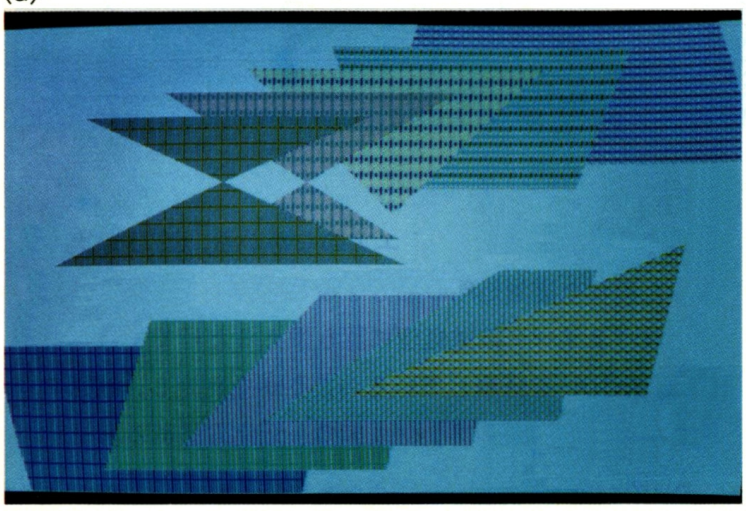
システムとしての性能向上をねらう

AGDC はデュアル・ポート・メモリの制御機能を備えている。また，CPU と組み合わせたシステムとしての性能を上げることを念頭に置いて設計した。内部はマルチプロセサ構成のアーキテクチャを採用している。AGDC 内には描画を実行する描画プロセサとは別に，プリプロセサという描画前処理用のプロセサをもたせた。

プリプロセサは X-Y 座標から絶対アドレスへのアドレス変換などの描画前処理を実行する。コマンド・インタフェースとしては CPU がプリプロセサの内蔵レジスタへコマンド・パラメータを直接書き込む方法を採用した。プリプロセサは描画を実行する描画プロセサとは独立に動作する。この結果，CGI や BASIC などの命令解釈や AGDC の駆動など，CPU が実行する描画前処理と，プリプロセサによる描画前処理，さらには描画プロセサによる描画実行とを，システム的にみて3段のパイプラインで処理することができる。

さらに，描画速度そのものを向上させるため，描画アルゴリズムの検討段階に，そのアルゴリズムに適した特殊な基本ハードウエアを設計した。描画プロセサは，これらのハードウエアを密に制御し，描画種類が変化しても応用が効くように設計している。ちなみに，1クロック 125 ns（8 MHz 時）で実行する各ステップで，最大6種類の命令実行および直接分岐を実行できる。各命令の実行にはパイプライン処理的な命令先取り機能をもたせている。したがって，この描面プロセサ自身も並列処理とパイプライン処理とを併せ持つ形態になっている。

(a)



(b)

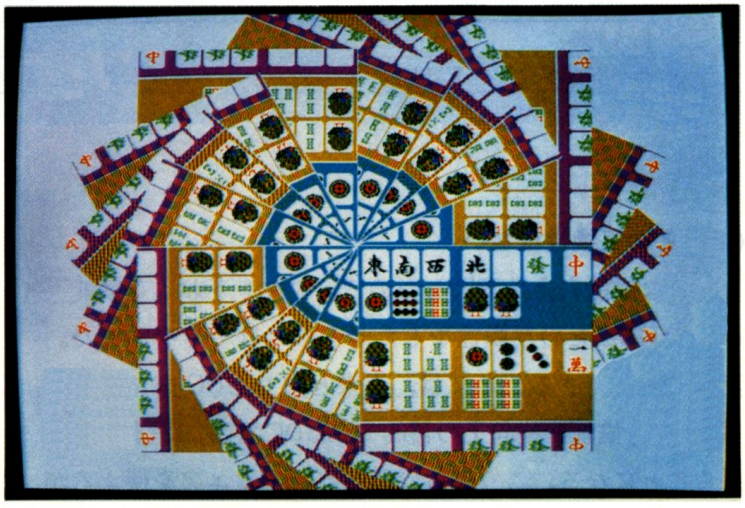


図1　描画例　640×480 ドットの表示画面を用いた。(a)は頂点の座標をパラメータとする一つのコマンドで台形を塗りつぶした。(b)は矩形領域の回転コピー結果である。倍率一定のまま回転すると1辺の長さが変わるため，回転角度に従って縮小処理を実行している。

表1　主なグラフィックス・コントローラの機能比較　機能としては備えているが実用上不十分と思われるものを"△"とした。機能の一部しかないときは"○"とした。ソフトウエアによってその機能を実現できるものについては"——"とした。

	HD63484 (日立製作所)	Am95C60 (Advanced Micro Devices, Inc.)	TMS34010 (Texas Instruments Inc.)	82786 (Intel Corp.)	μPD72120 (日本電気)
▷描画機能					
直線・四辺形	◎	○	○	◎	◎
円・円弧	◎	◎	——	◎	◎
楕円・楕円弧	◎	×	——	×	◎
扇形・弦形	×	×	——	×	◎
任意閉領域内塗りつぶし	△	◎	——	×	◎
四辺形内塗りつぶし	◎	◎	——	×	◎
円・楕円内塗りつぶし	×	◎	——	×	◎
三角形内塗りつぶし	×	◎	——	×	◎
台形内塗りつぶし	×	×	——	×	◎
▷表示メモリ内データ転送					
通常コピー	◎	◎	◎	◎	◎
90度回転コピー	◎	◎	——	◎	◎
傾斜コピ	×	×	——	×	◎
拡大・縮小コピー	×	○	——	×	◎
任意角回転を含む拡大・縮小コピー	×	×	——	×	◎
▷ハードウエア・クリッピング	◎	◎	×	◎	◎
▷プット・ゲット	×	◎	——	×	◎
▷描画位置指定	*X-Y* 座標	*X-Y* 座標	*X-Y* 座標	*X-Y* 座標 絶対アドレス	*X-Y* 座標 絶対アドレス
▷表示メモリ構成(最大容量)	ピクセル型 (2 M バイト)	プレーン型 (8 M バイト)	ピクセル型 (128 M バイト)	ピクセル型 (4 M バイト)	プレーン型, ピクセル型 (32 M バイト)
▷表示メモリのCPUへのマッピング	×	×	×	◎	◎
▷デュアル・ポート・メモリ制御	×	◎	◎	◎	◎
▷描画と描画前処理の分離	×	×	×	×	◎
▷表示と描画のクロック分離	×	◎	×	×	◎
▷表示機能					
画面分割	◎	×	×	◎	×
拡大表示	◎	×	×	◎	×
ハードウエア・ウインドウ	△	△	×	◎	×
カーソル出力	◎	×	×	◎	◎

GDCとの互換性はない

大幅な機能向上を意図したので，コマンドおよび機能レベルでGDCとの互換性をもたせることはできなかった。互換性を維持するためには，従来どおりGDCをマスタとして用い，描画機能向上の目的でAGDCを付加する構成を採ることになる。こうすれば既存ソフトウエアを使えるが，AGDCの機能を利用することはできない。しかし，AGDCを利用するルーチンを既存ソフトウエアに追加することは難しくない。既存ソフトウエアのなかから，CPUが実行していた複雑な描画処理の部分を抜き出し，AGDCに対する簡単なパラメータ転送処理に置き換えるだけですむ場合が多い。AGDCを搭載しているかどうかを判定し，制御フローを切り替えればソフトウエアの一元化も図れる。

AGDCは，パーソナル・コンピュータ，ワードプロセサ，ワークステーション，レーザ・プリンタなど，グラフィックス機能を必要とするすべての分野への応用をねらっている。

マルチプロセサ構成の内部アーキテクチャ

AGDCの内部は以下に示す5種類の内部機能ブロックに分かれている（**図2**）。

① プリプロセサ
② 描画プロセサ
③ 表示プロセサ
④ 同期信号発生器
⑤ CPUインタフェース

描画，表示，前処理用に3種のプロセサを搭載するマル

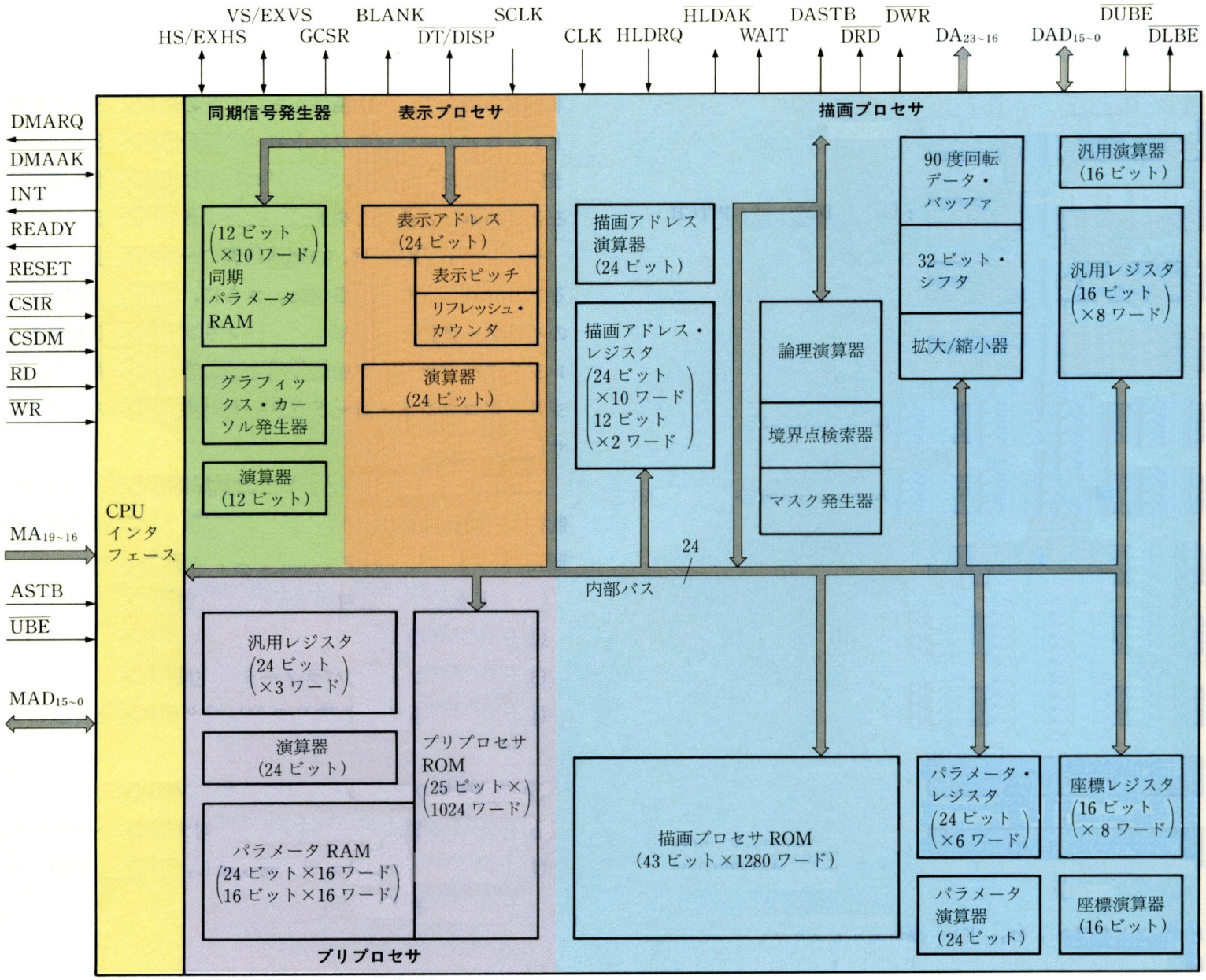


図2　内部構成　同期信号発生器，表示プロセサ，プリプロセサ，描画プロセサ，CPUインタフェースという5種類の機能ブロックで構成する。各ブロック間は24ビットの内部バスで結ぶ。アーキテクチャの特徴は，プリプロセサと描画プロセサでパイプライン処理を行なうこと，描画プロセサ内に高速の塗りつぶしや拡大・縮小を行なうハードウエアをもたせたことである。

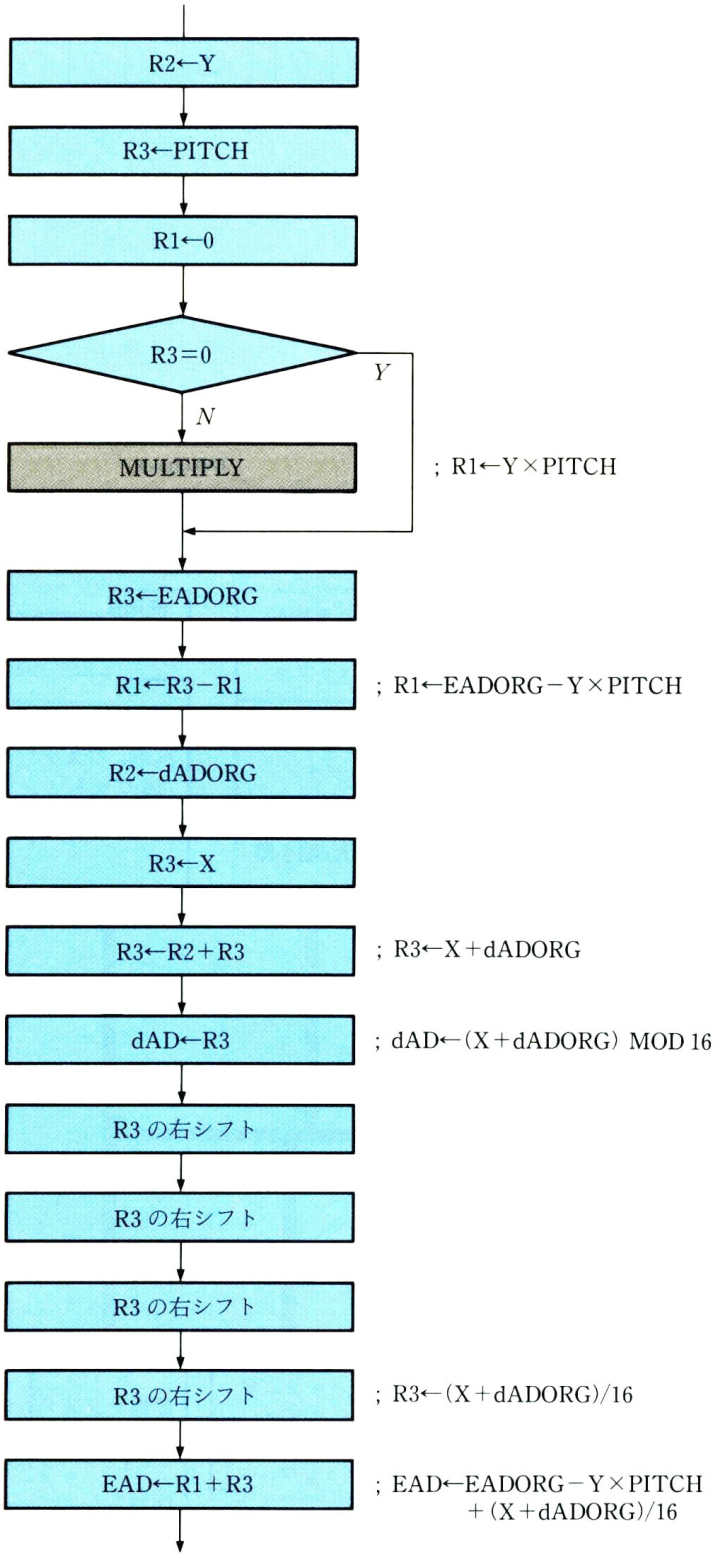


図３　X-Y 座標から絶対アドレスを計算するプリプロセサの処理手順　一つのステップを 1 クロック(8 MHz 時には 125 ns) で実行できる。プリプロセサ内の加減算器で図中に示した MULTIPLY という乗算を実行する。プリプロセサを搭載したことで，描画前処理は CPU で処理する場合より 10 倍程度速くなった。

チプロセサ構成を採用した点がアーキテクチャの特徴である。この構成により，CPU，プリプロセサ，描画プロセサというパイプライン処理を可能にした。また，描画プロセサには描画速度を上げるための専用ハードウエアをもたせている。

プリプロセサで描画の前処理を実行

GDC は CPU からコマンドを受け取るための FIFO を内蔵していた。主に描画速度と前処理速度の違いを調整するためである。しかし，この FIFO にはコマンド・パラメータを一時的に記憶する受動的な役割しかない。一方，集積度の向上で，*X-Y* 座標からメモリの絶対アドレスを算出するなどの前処理をコントローラに分担させる方向が一般的になった。ただし，描画プロセサにこの機能を追加すると，前処理と描画とを並列に実行できなくなる。

そこで，AGDC では，描画前処理の一部を高速に実行するプリプロセサを内蔵させた。さらに，受動的な FIFO へのパラメータ書き込みではなく，プリプロセサの管理するレジスタへパラメータを直接書き込む方式を採用した。レジスタにコマンド・コードを書いた時点から，プリプロセサは動作を開始する。

プリプロセサは，24 ビットの加減算器，汎用レジスタ，制御 ROM，パラメータ RAM などで構成している(図 2 参照)。プリプロセサの主な機能を以下に示す。

① *X-Y* 座標から絶対アドレスへの変換
② コマンドの解釈
③ 図形描画に必要なパラメータの生成
④ *Y* 座標を基にした塗りつぶしパターン抽出位置の算出
⑤ 三角形塗りつぶしのための座標のソーティング
⑥ ユーザの設定したパラメータに対するエラー・チェック
⑦ 描画プロセサとのデータ受け渡し，および，描画プロセサの起動制御

X-Y 座標から物理アドレスを高速に計算

プリプロセサは *X-Y* 座標から表示メモリの絶対アドレスを計算する。次の計算式を用いると座標(*X-Y*)から絶対

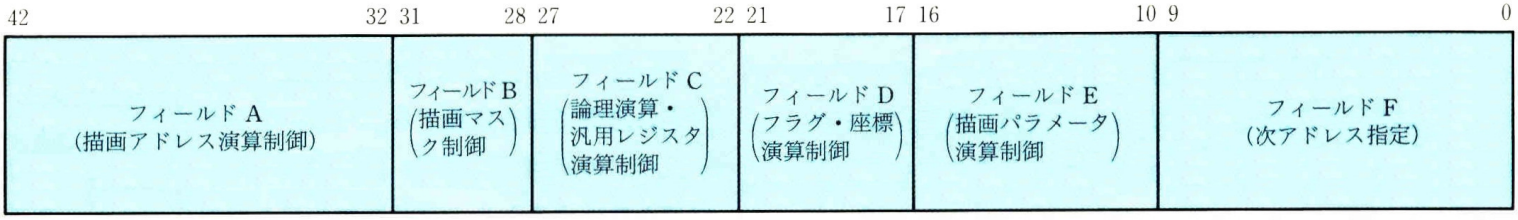


図4　描画プロセサの命令形式　命令は43ビット長で，AからFまでの6種類のフィールドから成る。各フィールドの命令は他のフィールドと独立して実行できる。演算器やシフタの数はフィールド数よりも多いので，一つのフィールドで複数の演算を制御できる。

ワード・アドレスEAD，ドット・アドレスdADを算出できる。

$$EAD = EADORG - Y \times PITCH + \{(X + dADORG)/16\} \quad (1)$$

$$dAD = (X + dADORG) \ MOD \ 16 \quad (2)$$

EADORGとdADORGはX-Y座標の原点(0,0)に相当する絶対アドレス（それぞれワード・アドレスとドット・アドレス）を示す。PITCHは表示メモリの水平方向のワード数である。

プリプロセサはこの計算式に従うアドレス計算を最小のハードウエアで高速に実行する。そのために既存の汎用プロセサを流用するという安易な方法は採らなかった。処理の高速化や無駄を省いた回路構成とするには，最適化されたプロセサを新たに設計する必要があったからである。プリプロセサの24ビット加減算器を用いると，たとえば，乗数と被乗数が16ビットで積が24ビットの乗算を，1ビット当たり1クロックで実行できる。この速度を実現するために，3本の演算レジスタ中の1本に右シフタ，他の1本に左シフタを付加した。また加減算器に演算終了判定回路を組み込んだ。このハードウエアは整数除算も高速に実行できる。

アドレス計算の流れを**図3**に示す。図中のMULTIPLY (Y×PITCH) というのが加減算器による乗算を示している。表示画面の水平解像度が1024画素のとき，そのワード数を表すPITCHは64（実質7ビット）である。したがって，この乗算は7クロックで実行できる。X-Y座標から物理アドレスへの変換は，この7クロックに，他のステップ（図3参照）を加えた22クロックで実行することになる。

描画プロセサに
拡大・縮小，境界点検索などを行なうハードウエアを搭載

プリプロセサは一つの演算器しか備えていない。一方，描画プロセサは描画を高速化するために複数の演算器やシフタ，マスク生成器などを備えている。これらの回路は一つのサイクル内で同時に制御できるようにしてある。命令は43ビット幅で，6種のフィールドから成る。その各フィールドに異なるハードウエアの制御を受け持たせた（**図4**）。最大2種類の判定結果によるアドレスの4分岐も，一つのサイクル内に実行できる。したがって，アドレス分岐や演算命令などをすべて1クロック（125 ns）で実行できる。たとえば，24ビットのレジスタ読み出し，演算，再書き込み，演算結果の判定は1クロックで終了する。

この処理速度を達成するために，制御記憶を高速にアクセスする工夫を施した。非同期型のROMを用いることで，アドレスを与えればすぐに出力が得られるようにした。しかし，アドレス・サイクルに同期して命令を取り出すために，すべての命令デコード出力ドライバに同期型一時記憶素子を付加する回路構成を採った。これにより制御ROMはすべて1サイクルでアクセスできる。

描画プロセサの24ビットの描画アドレス演算器，描画パラメータ演算器，16ビットの座標演算器，汎用演算器などにはそれぞれ密に接続したレジスタ群をもたせている。このほかにも表示データ論理演算器，90度回転データ・バッファ，32ビット・シフタ，データ拡大/縮小回路，描画マスク発生器，境界点検索器などのハードウエアを満載している（図2参照）。

傾斜コピーでは1本の直線発生器，三角形や台形フィル，任意角回転コピーでは2本の直線発生器，円や楕円フィル

では1個の円・楕円発生器を使用して必要な座標を発生させている。こうした座標発生器を固定機能のハード・ワイヤド・ロジックで組み込んだわけではない。描画プロセサはソフトウエア変更だけで，共通なハードウエアをさまざまな用途に効率よく使えるようにしてある。たとえば，直線発生器はパラメータ・レジスタ3本とワーキング・レジスタ1本で構成している。

また，AGDCはピクセルまたはドット単位で描画する機能と，ワード単位で描画する機能を併せ持っている。異なる描画に対しても不都合なく動作するように描画マスク発生器を新たに開発した。このなかにクリッピング制御器も含めている。マスク発生，クリッピング制御は他の描画処理と並列に動作する。したがって，クリッピング動作を指示しても描画速度は遅くならない。

複雑な描画は分割し，プリプロセサと描画プロセサでパイプライン処理

座標レジスタや描画用アドレス・レジスタなどは描画プロセサとプリプロセサの両方にもたせた。このため，プリプロセサが処理を終了している状態であればプリプロセサのレジスタ内容を更新しても描画動作には影響しない。プリプロセサと描画プロセサは各々の処理を独立に実行できることになる。その結果，描画プロセサが描画実行中に，プリプロセサは次の描画のための前処理を実行できる。

プリプロセサには描画プロセサの状態検出機能に加え，起動指示機能ももたせた。したがって，一つの描画コマンドであっても，描画プロセサの動作を何回かに分けて指示することができる。たとえば，円弧扇形描画の場合，実際は次のように処理を進めている。

① 円弧の描画パラメータ算出（プリプロセサ）。
② 円弧の描画（描画プロセサ）。
③ 描画の開始点および終了点を描画プロセサから読み出した後，終了点と中心間の直線描画パラメータ算出（プリプロセサ）。
④ 終了点と中心間の直線描画（描画プロセサ）。
中心-開始点間の直線描画パラメータ算出（プリプロセサ）。
⑤ 中心-開始点間の直線描画（描画プロセサ）。

AGDCでは，円弧や楕円弧を描画する場合，開始点と終了点の座標で弧を定義する。一方，ビジネス・グラフィックスなどの応用では普通，角度で弧を定義する。このとき，座標を算出するための三角関数演算を簡略化すると，誤差のために実際に描画する弧には存在しない座標を指定してしまうことが起こる。AGDCではこのような設定がなされても座標に補正を加えて描画できるように考慮した。

第1象限に存在する弧を例に考えてみよう（**図5**）。接線の傾きが135度より小さいときには，ユーザの設定した描画開始点が実際の弧の上に存在していなくとも，Y座標だけが一致していればよい。135度より大きいときにはX座標だけが一致していればよい。135度のときには，XとY両方の座標を用いた大小比較（第1象限の場合は，描画位置がユーザの設定した位置より小さいかどうか）を判定している。この結果，座標をどの位置に設定したとしても描画の開始点または終了点を補正できる。このため，かなり乱暴な座標指定をしても暴走してしまうことはない。前述したプリプロセサ，描画プロセサの処理の流れでは，正確な座標値を③の処理によってプリプロセサが読み取っている。

ハードウエア・ウインドウは採用せず，表示プロセサを単純に

描画プロセサ，プリプロセサ以外の部分を説明しよう。AGDCでは，GDCでの応用実績をみて表示プロセサから拡大表示機能や画面分割機能を削除した。これらの機能を用いれば，表示アドレスの変更だけで高速に画面を分割したりスクロールできる。ところが，スクロール範囲や分割方法は実装する表示メモリの大きさに依存してしまうなど，ソフトウエアの汎用性や機能に制約が生じる点が問題だった。ハードウエア・ウインドウ機能を搭載しなかった理由の一部もこの点にある。この結果，表示プロセサ部分は簡素化できた。マルチウインドウ表示は描画プロセサのコピー機能を用いて実現できる。また，表示プロセサはダイナミックRAMのリフレッシュ機能を備える。リフレッシュ・サイクル時には，13ビットのリフレッシュ・アドレスを

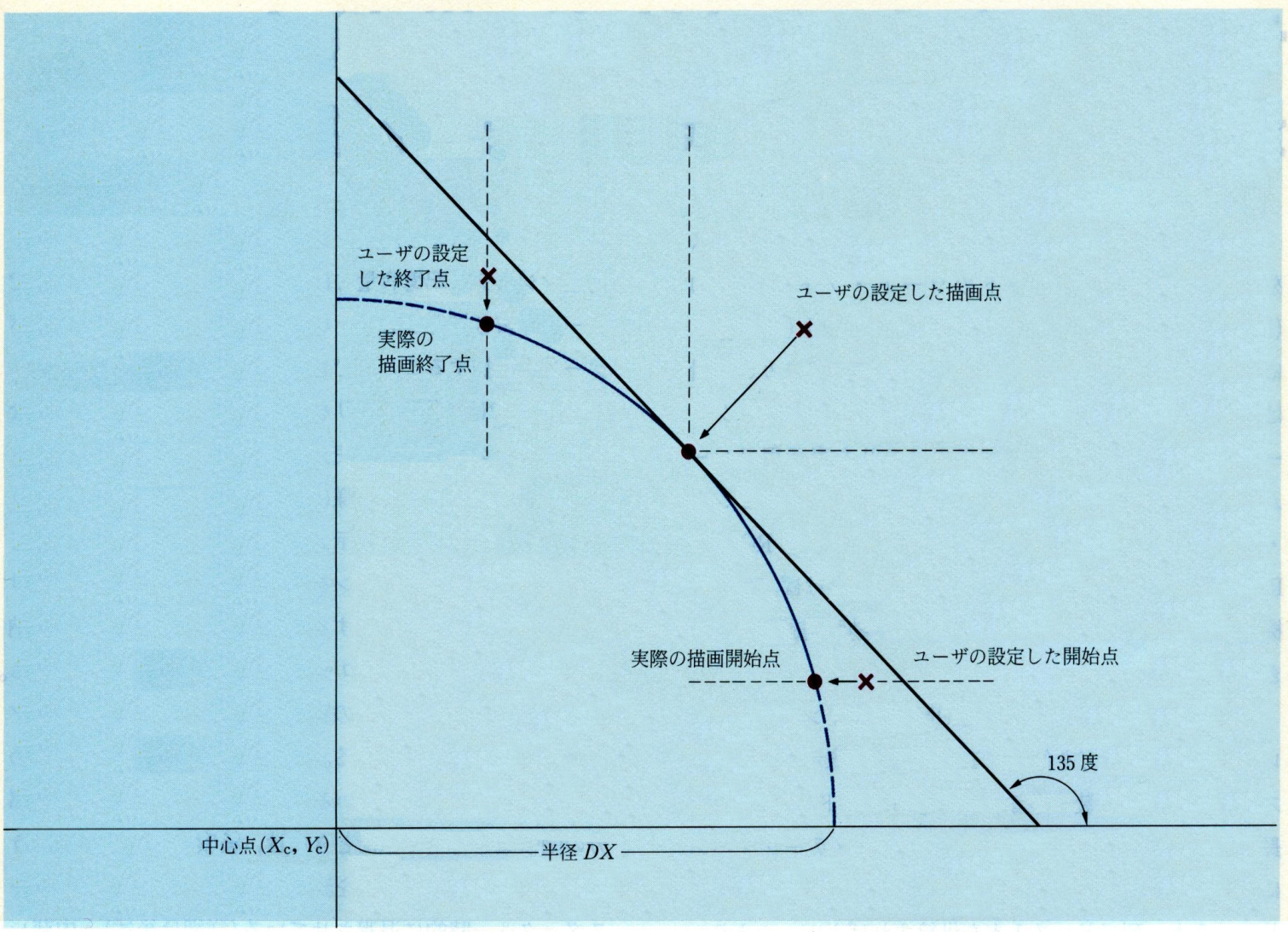


図5　弧の描画　AGDCでは描画の開始点，終了点を指定することで弧を描画する。ユーザの指定した点が実際の弧の上に存在しない場合は，描画点を補正できる。

表示メモリ・アドレス出力の下位ビットに出力する。

同期信号発生器は，あらかじめユーザの設定したパラメータ値に従って水平/垂直の同期信号(HS/VS)や表示消去信号(BLANK)などを出力する。また，表示メモリのアドレスに依存しないグラフィックス・カーソル表示用のタイミング出力(GCSR)をもつ。カーソルは表示画面の左上を原点とした任意の位置に表示できる。カーソルの形状はクロスヘアまたはブロックである。

CPUインタフェースは，システム・バスとAGDC間の信号授受のタイミングを制御する。CPUが表示メモリをアクセスするときに動作するアドレス拡張や表示メモリ・バスのアービトレーション機能も有する。

AGDCはCLKとSCLKと呼ぶ2系統のクロック入力端子をもつ。同期信号発生器と表示プロセサには，SCLKから入力したクロックを供給している。SCLKの周波数は使用するディスプレイ装置の仕様や表示分解能などに依存して決まる。したがって，クロック周波数をコントローラの最高動作周波数(f_{max})よりも低く設定しなければならないことも起こり得る。その場合には，コントローラの備える描画速度を100%発揮することはできない。そこで，AGDCでは，プリプロセサと描画プロセサにはCLKというSCLKとは別系統のクロックを供給できるようにした。この結果，プリプロセサと描画プロセサは常に最高動作周波数で動作できる。

デュアル・ポート・メモリなどで構成した最大32Mバイトの表示メモリを制御できる

AGDCはシステム・バスから分離した表示メモリ・バスを直接制御する。24ビットのワード・アドレス出力($DA_{23\sim16}$および$DAD_{15\sim0}$)とその下位16ビット($DAD_{15\sim0}$)に入出力するデータ端子をもっている。表示メモリの容量は最大32Mバイトで，1024×1024ビットのメモリ・プレーン256枚に相当する。すべてのメモリを表示用フレーム・バッファとして割り当てなくてもよい。漢字フォント・パターンや動画用パターンを格納しているROMやRAMを表示メモリ領域の一部に実装しておけば，コピー機能を使って高速な文字フォント展開や動画制御などができる。

漢字フォント・メモリ（漢字ROM）はライン方向のアドレス・カウンタなどを付加することなく，表示メモリ・バス上にそのまま実装できる。漢字JISコードは14ビット長である。また，16ビットの表示メモリ・データ・バスに，たとえば32×32ドットのデータを出力するには，行アドレス5本，列アドレス1本を供給すればよい。つまり，漢字コード用14ビット，フォント用6ビットの，計20ビットのワード・アドレスがあれば，32×32ドット構成の第1，第2水準の漢字ROMを実装できる（AGDCはワード・アドレス24ビット)。

バイト・アドレスが20本の16ビットCPUでは，32×32ドットの漢字ROMを主記憶上に直接マップすることすらできない。システム・バス側に文字フォント・メモリを実装し，フォント・データをコントローラに転送するようなシステム構成を採ると，フォント・データの読み出しやその転送動作だけで多くの時間を浪費することになる。

表示メモリを塗りつぶしパターンの格納やスタックとして使用

多くのコントローラでは，描画を実行するたびに，必要とする塗りつぶしパターンや文字フォント・データなどをシステム・バスから内蔵レジスタに設定する。ところが，内蔵レジスタの記憶容量には上限があるため，描画途中でデータの書き換えが必要となったり，複雑な塗りつぶしパターンを選択することができなかった。

AGDCでは，表示メモリを記憶容量の制約がほとんどないワーキング領域として使えるようにした。文字フォントや動画用のパターンと同様に，塗りつぶしパターンも塗りつぶしを実行する前に表示メモリにあらかじめ記憶させておく。塗りつぶしパターンを表示メモリ上に格納しておけば，そのパターンを転送する必要はない。主記憶上に格納した場合でもCPUのブロック転送命令一つで転送できる。

表示メモリはペイントの実行に必要なデータのスタック領域としても使用している。塗りつぶす領域を決めるために境界点検索を実行しているとする。この間に，ある一部分の閉領域を特定できたときには，検索を中断してしまうのではなく，その閉領域を表現するために必要なデータをスタックへ一時的に退避させている(詳細は後述)。内蔵レジスタをデータ・スタックのために使用してもよいが，この場合にもレジスタの記憶容量の点からスタック・レベルの上限に制約が生じてくる。境界点検索のアルゴリズムにも依存するが，複雑な図形になるにつれてスタック・レベルがその上限を超える確率は高くなる。

表示メモリ内にワーキング領域を設定するには，その先頭絶対アドレスとその大きさをAGDCに与える。定義したスタック領域がオーバフローしたときには，動作を停止し描画プロセサ・エラーであることをCPUに知らせる。

CPUは表示メモリおよびAGDCの内部レジスタを直接アクセスできる

CPUが表示メモリを主記憶と同等にアクセスするには，表示メモリをCPUのメモリ・アドレス上に主記憶の一部としてマップしなければならない。このため，AGDCは，

システム・バス（CPU）のバイト・アドレス20本（$MA_{19\sim16}$と$MAD_{15\sim0}$）を入力するアドレス端子と，CPUが表示メモリをアクセスするときに低レベルにする端子（$\overline{CSDM}$）とを備えている。

表示メモリ・バスのアドレスは24ビットのワード・アドレスである。CPUのバイト・アドレスを24ビットのワード・アドレスに拡張するためAGDC内に8ビットのバンク・レジスタを設けた。**図6**にアドレス拡張方式を示す。バンク・レジスタに20H（図では　）を設定しておき，24DA8Hまたは24DA9H（図では　）のバイト・アドレスを与えたときの拡張例を示してある。　で示した表示メモリ下位バイト・エネーブル信号（$\overline{DLBE}$）はバイト・アドレス最下位ビットと等しい。システム側のデータ・バス幅は8ビットでもかまわない。このときは，　で示した上位バイト・エネーブル信号（$\overline{UBE}$）を常に高レベルにして使用する。

座標レジスタなどのAGDCの内部レジスタをCPUからアクセスすることもできる。AGDCのパラメータ・レジスタ類は，CPUのバイト・アドレス下位8ビットによって選択できる。チップ・セレクト信号（$\overline{CSIR}$）を低レベルにすると内蔵レジスタがアクセス可能になる。このようにAGDCの制御下にある表示メモリと内蔵レジスタをCPUによって直接アドレスできるため，汎用的なソフトウエア・デバガを用いて簡単にそれらの内容をダンプしたり書き換えたりできる。

バス・アービタを内蔵

こうしたマップ機能に関連して，表示メモリ・バスの使用優先順位づけを制御するバス・アービタを内蔵している。また，CPUに対してウエイト動作を要求するレディ信号（READY）を出力する機能もある。したがって，タイミング設計が難しいバス・アービタを外部回路で構成する必要はない。表示メモリ・バスの使用優先順位を高い順に並べると以下のようになる。

① AGDCによる表示アドレスおよびリフレッシュ・アドレスの発生

② 表示メモリ・バスの使用許可を受けた他のプロセサに

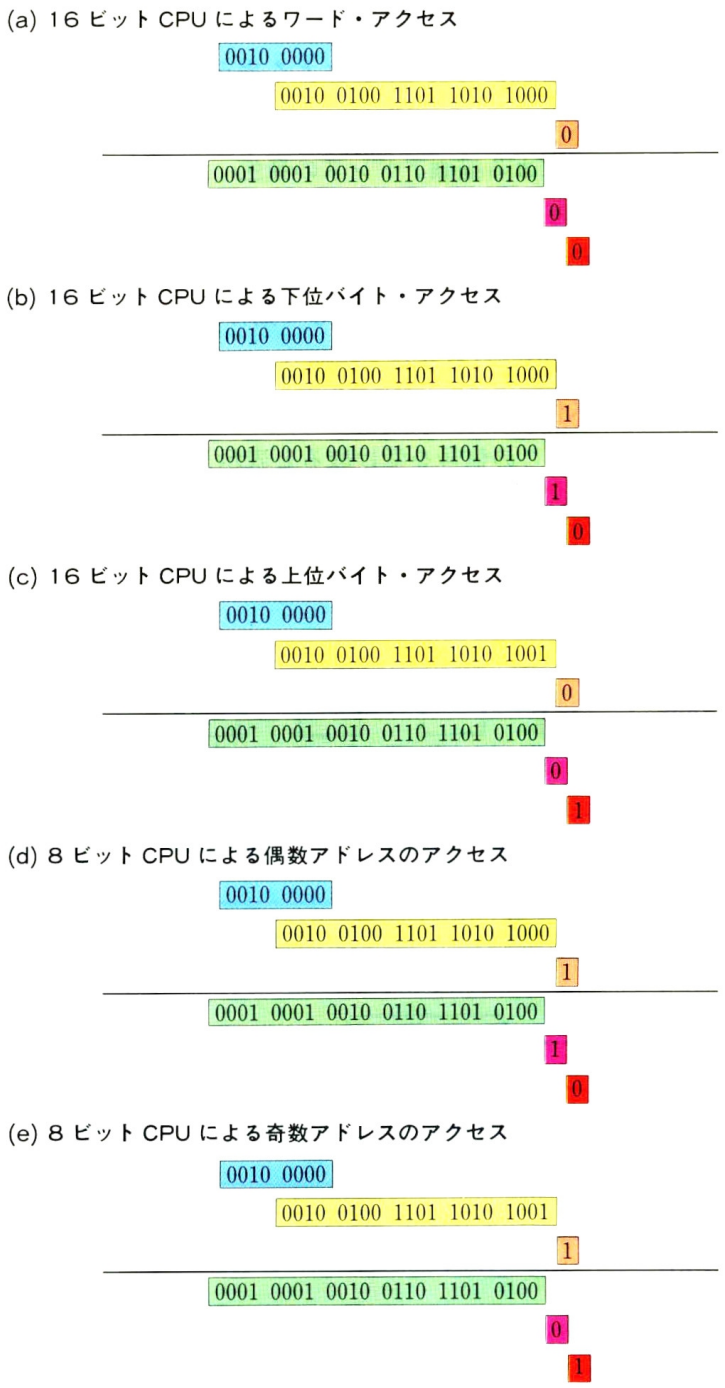


図6　CPUによる表示メモリ・アクセス方法　　は内蔵バンク・レジスタ内容，　はシステム・バスから入力したバイト・アドレス，　はシステム・バスの上位バイト・アクセス時に低レベルとなる$\overline{UBE}$信号，　はAGDCが出力する表示メモリ・ワード・アドレス，　は表示メモリ・バスの上位バイト・アクセス時に低レベルとなる$\overline{DUBE}$信号，　は下位バイト・アクセス時に低レベルとなる$\overline{DLBE}$信号を示す。

16ビットのデータ・バス幅をもつCPUによるワード・アクセス，下位バイト・アクセス，上位バイト・アクセス，および，8ビットCPUによるバイト・アクセスに対応したアドレス拡張ができる。

よる表示メモリ・アクセス

③ AGDC を経由した CPU による表示メモリの直接アクセス

④ AGDC による描画アドレスの発生

AGDC は優先順位の高い表示アドレスの発生やリフレッシュ・アドレスの発生を行なう。このため，通常は表示メモリ・バスのバス・マスタとして動作する。表示メモリ・バスを外部のプロセサなどが使うことを考慮して，表示メモリ・バス使用権要求信号の入力端子（HLDRQ）を設けた。AGDC はこの信号を受け取ると表示メモリ・バス側のアドレスとデータ出力をフローティング状態にするとともに，描画を実行中である場合にはその描画を一時的に停止させる。さらに，承認信号の出力端子（$\overline{\text{HLDAK}}$）の立ち下がりによって要求受け付けを要求源に示す。また，AGDC による表示やリフレッシュのためのアドレス供給は優先順位が一番高いので，$\overline{\text{HLDAK}}$ の立ち上がりによってバス使用権の返還を逆に要求する。

4 種類のアドレス・サイクルを用意

AGDC は次の 4 種類の表示メモリ・バス・アドレス・サイクルをもつ（**図 7**）。

① 描画アドレス・リード・サイクル

② 描画アドレス・ライト・サイクル

③ 表示アドレス・サイクル

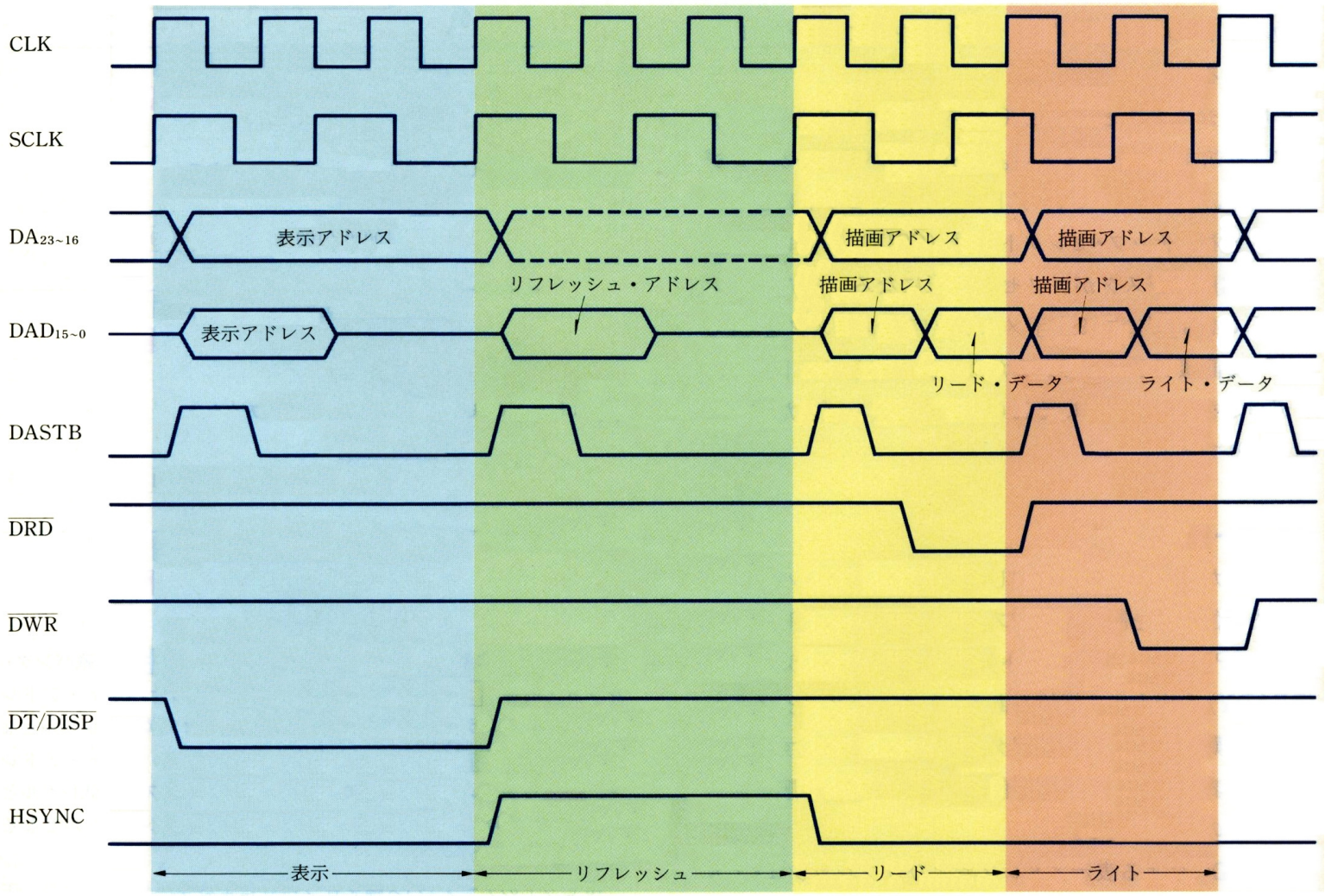


図 7　表示メモリ・アドレス・サイクル　表示メモリをアクセスするときには，必ずアドレス・ストローブ信号（DASTB）を出力する。アドレス・ストローブ信号は，表示およびリフレッシュ時には SCLK に同期し，リードおよびライト・サイクル時には CLK に同期している。

④ リフレッシュ・アドレス・サイクル

GDCの描画アドレス・サイクルは常にリード・モディファイ・ライトを実行するため4クロックであった。AGDCでは，描画サイクルはCLKに同期して2クロック，表示とリフレッシュ・サイクルはSCLKに同期して2クロックにしてある。ダイナミックRAMに対するリフレッシュ動作の実行を選択すると，水平同期期間にリフレッシュ・アドレス・サイクルを挿入する。描画アドレス・サイクルでは，2クロックごとに連続して読み出しや書き込み動作を実行することもあれば，描画サイクルと次の描画サイクルとの間に1クロック間の空き時間を挿入することもある。

各アドレス・サイクルの先頭では，必ずアドレス・ストローブ信号(DASTB)を出力するので，外部回路はそのアドレス・サイクルを正確に把握できる。このDASTBはアドレス・データ多重化端子からアドレスだけを抜き出したり，ダイナミックRAMに供給するRAS信号，CAS信号を生成するための基準タイミング信号として使用する。

描画アドレス・リード・サイクルでは，読み出し信号($\overline{\text{DRD}}$)を出力するので，データ・バスの方向制御が行なえる。描画アドレス・ライト・サイクルでは，書き込み信号($\overline{\text{DWR}}$)を出力する。これらの信号は，外付け回路によって各々の装置に適合した遅延を加えた後，表示メモリに供給する。表示サイクルとリフレッシュ・サイクル時には，DASTBしか出力しない。表示サイクルであることは，データ転送タイミング信号と表示サイクル・タイミング信号との兼用出力端子($\overline{\text{DT}}/\overline{\text{DISP}}$)の信号により判別できる。リフレッシュ・サイクルであることは水平同期信号(HSYNC)でわかる。

デュアル・ポート・メモリにデータ転送信号を出力

デュアル・ポート・メモリの使用を考慮して，次の2種の描画タイミング動作モードを用意した。

① データ・トランスファ・モード

② メモリ・サイクル・スチール・モード

表示メモリとしてデュアル・ポート・メモリを使用する場合には，データ・トランスファ・モードを選択する。このモードでは，異なった周波数のクロックをCLK端子，SCLK端子に供給してもよい。ただし，異なるクロックに同期した信号の同期をとるのに必要な同期化回路の構成から，

f_{max}(最大動作周波数)≧CLK≧SCLK≧CLK/2

の関係を満たさなければならない。この状態ではリフレッシュ・サイクルとデータ転送サイクルには描画できない。CLKを単位として，さらにその前2クロック，後1クロックの間は描画できなくなる場合がある(**図8**(a))。

CLKとSCLKに同一のクロックを供給した場合には，フラグの設定によって，同期化回路をバイパスできる。この状態では，リフレッシュ・サイクルとデータ転送サイクル以外のすべてのタイミングで描画を実行できる(図8(b))。

描画タイミング選択をデータ・トランスファ・モードにすると，デュアル・ポート・メモリに対してデータ転送タイミング信号($\overline{\text{DT}}$)を出力する。次の2種の$\overline{\text{DT}}$信号を発生するタイミングが選択できる。

① 1走査線の表示開始時および表示アドレスの下位8ビットがゼロになったとき。

② 1画面表示の開始時および表示アドレスの下位8ビットがゼロになったとき。

表示メモリの水平方向を表示画面よりも大きく定義し，水平方向へのスクロールを可能にした構成の場合には，①を選択する。表示メモリの水平方向が表示画面と同じで，ノンインタレース走査を行なっている場合には，画面に向かって右端と次の走査線の左端の表示アドレスには連続性が保たれているので②を選択できる。②は①よりも$\overline{\text{DT}}$信号発生の頻度が低いのでその分，描画可能タイミングが増加する。デュアル・ポート・メモリを使用すれば，ドット周波数が数百MHzとなるような高解像度の表示装置にも対応できる。

表示画面が高解像度になるにつれ，単位時間当たりの表示データの読み出し量は多くなる。このため，$\overline{\text{DT}}$信号の発生間隔を短くする必要が生じてくる。AGDCでは，高いクロック周波数を維持しつつ低解像度から高解像度表示に対応するため，表示アドレスのインクリメント形態を《1/4, 1/2, 1, 2, 4, 8, 16, 32》の8種用意した。概念的に言えば，640×400ドットのノンインタレース表示であれば《1》

が適用できる。これよりも低解像度であれば小さい値，高解像度になるにつれて大きな値を選択することになる。

メモリ・サイクル・スチールもできる

コストをできるだけ低く抑えたい場合やレーザ・プリンタのように表示を特に必要としない場合には，デュアル・ポート・メモリではなく通常のRAMを表示メモリとして使用することもできる。このため，メモリ・サイクル・スチール・モードを設けた。このモードでは，CLKとSCLKには同一信号源のクロックを供給しなければならない。

リフレッシュおよび表示期間中では，リフレッシュ・アドレス発生を含む表示サイクルと描画を実行できる描画サイクルとを2クロックごとに交互に切り替えて発生させている(図8(c))。このため，この期間での描画実行可能なタイミングは単位時間当たり50%になる。表示期間以外では，リフレッシュ・サイクルを除きすべてのタイミングで描画を実行できるので，この値は少しだけ増える。ただし，メモリのサイクル時間の制約から解像度の上限が決定されるので，このサイクル・スチール・モードは比較的解像度の低い装置にしか適用できない。水平走査周波数が24 kHzの標準的なディスプレイを使用し，表示メモリのデータ・バス幅を16ビットとし，ダイナミックRAMのサイクル時間を300 nsとすれば，640×400ドットのノンインタレース表示あたりにその上限がくる。

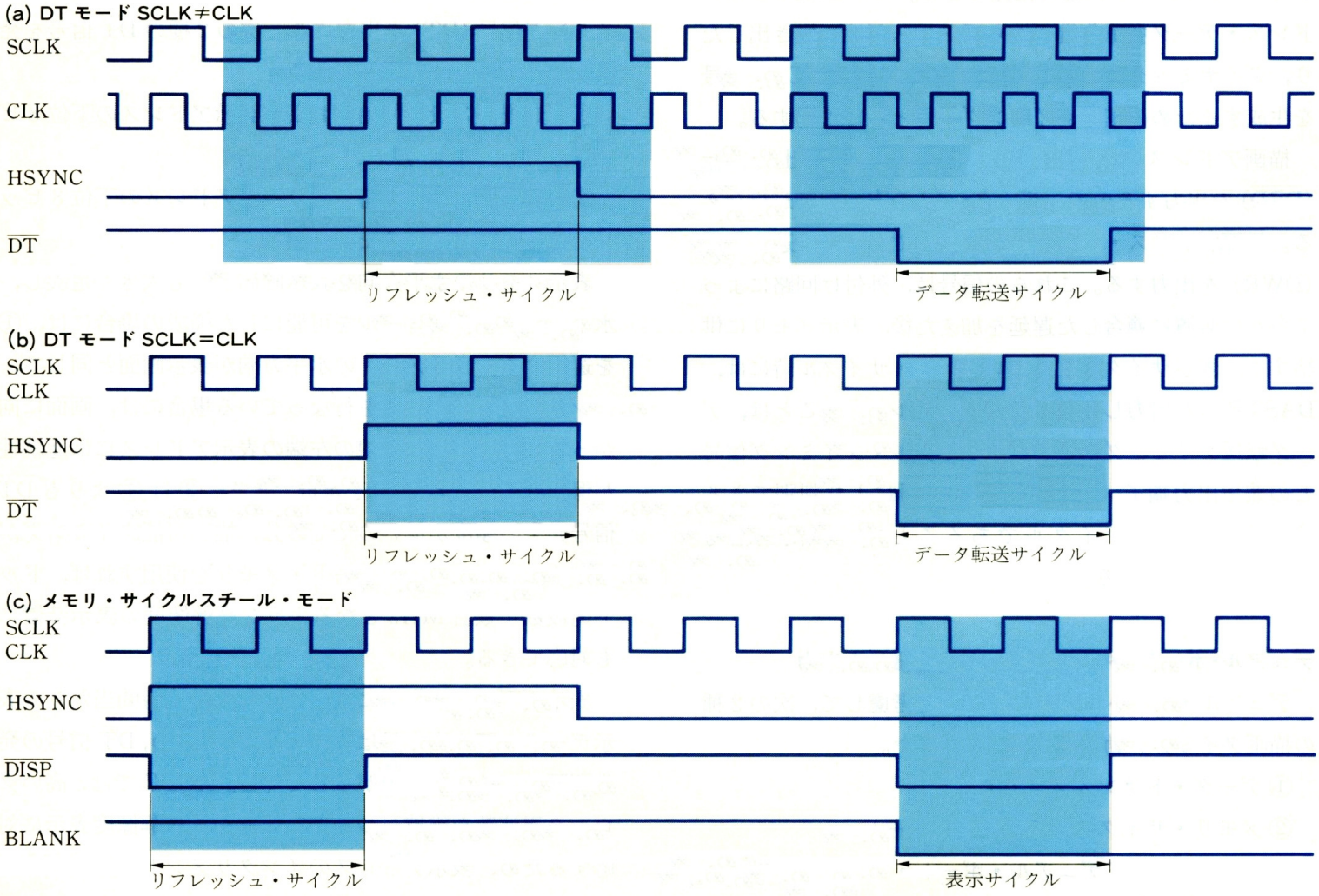


図8　描画のできないタイミング　■色で示す期間には描画は実行できない。クロック発生源の異なるDTモード(SCLK ≠ CLK)，同一のクロックを供給したときのDTモード(SCLK=CLK)，サイクル・スチール・モードのときで描画できないタイミングは変化する。

パラメータ直接転送方式のコマンド・インタフェースを採用

描画プロセサに高速描画機能をもたせても，それをいかに効率良くシステム・レベルで生かせるかが重要である。つまり，CPUとコントローラでコマンドを受け渡す手順（コマンド・インタフェース）によって実際の性能は左右される。われわれは開発の初期からこの方法を十分に検討した。

コマンド・インタフェース法は全体像としてとらえると次のように区分できる。

① システム・バス側でのデータ転送方法
　(a)パラメータ直接転送
　(b)パラメータ間接転送
② コントローラの状態検出方法
　(a)ステータス・フラグ
　(b)割り込み要求
　(c)ウエイト要求
　(d)DMA要求
　(e)ホールド要求
③ 転送データの格納形式
　(a)間接アドレス指定
　(b)直接アドレス指定
④ 転送データの格納先
　(a)レジスタ
　(b)FIFO

AGDCは，①(a)，②(a)，(b)，(c)，③(b)，④(a)の組み合わせを採ることにした。

パラメータ間接転送方式(①(b))はデータ生成時間が全体の描画時間を引き延ばすため採用しなかった(後述)。この方式と密着したDMA要求やホールド要求（②(d)，(e)）も除外した。転送データの格納形式に関しては，直接アドレス指定を採用した。間接アドレス指定は，コマンドとパラメータまたはアドレスとデータとを区別して転送データを組み立てねばならない。また，アドレスとデータとを同時に転送できる直接アドレス指定に比べて明らかに転送バイト数が増加するためである。転送データの格納先はレジスタにした。FIFOを用いても記憶容量に上限が生じる。また，あらためて読み出し動作が必要になるためである。

AGDC内のレジスタはCPUのアドレス空間にマップされている。このため，CPUがコマンドをAGDCに与えるには，主記憶に対する通常のデータ・ムーブ命令を実行するだけでよい(**図9**)。図9で　　の部分がAGDC内のパラメータRAMへの書き込みに対応している。パラメータRAM内の32ワードのレジスタはそれぞれのアドレスに対応して役割が決められている。プリプロセサはコマンドが書き込まれた時点で描画前処理を開始する。

状態検出方法は3種類用意した（②(a)，(b)，(c)）。これらの方法を描画種類によって切り替えながら使用してもよい。パラメータ生成に専念できる専用のCPUとAGDCでシステムを構成した場合，ウエイト要求（②(c)）によるハードウエア・ハンドシェイクが最適である。CPUはAGDCに対する状態検出をする必要はなく，AGDCに対するアクセスを主記憶に対するアクセスと同等に扱える。しかし，CPUがウエイト状態のときには割り込み処理は実行できない。ステータス・フラグ（②(a)）によるソフトウエア・ハンドシェイクを選択すれば，コントローラがビジーのためパラメータ類を転送できない状態でCPUがループしていたとしても割り込み処理は実行できる。

どのような状態検出方式を採った場合でも，ウエイト制御信号（READY）はCPUのウエイト制御を行なう端子に接続しなければならない。READY信号の発生期間は種々の条件下において異なる。CPUが表示メモリをアクセスする場合を例に採ると，読み出し時にはシステム・バスの読み出し信号の立ち下がりからウエイト信号が5クロック期間だけ発生する。書き込み時にはウエイト信号を発生しない。

パラメータ間接転送方式には問題がある

パラメータ間接転送方式には，DMA転送によってコマンド・パラメータを受け取る方法や主記憶上にあらかじめ作成しておいたコマンド・パラメータ・リストをコントローラ自身が読み取りにくい方法（リンクト・リスト）などがある。この方式では，座標列などの原型データから，コントローラに依存するコマンド・コードや転送アドレス情報などを含む転送データ列を生成する。このため，座標データを生成したらすぐに転送してしまうパラメータ直接転送方式に比べると，そのデータ生成処理が途中に入ることになる。その分，CPUの稼働時間は増え，パラメータ転送用のソフトウエアは複雑になる。

多数の直線描画で文字を描画するストローク文字描画のように，一連の描画処理の見通しが明瞭な場合には，コントローラの状態に依存してCPUの描画前処理を中断することがない間接転送方式の利点を生かせるかもしれない。しかし，DMA転送方式では転送するデータ・ブロックの大きさをどのようにして最適化するかが大問題である。転送ブロックの大きさを固定することには基本的に無理がある。データがそろうまで転送できなくなってしまうからである。また，転送ブロックを細分化するにつれ，転送データの生成やDMAコントローラの動作設定などに要する処理時間は増加し，全体の描画速度は低下してしまう。

コントローラ自身がコマンド・リストを読みにいくリンクト・リスト方式ならば，このような問題は生じない。しかし，制御ソフトウエアはDMA転送方式以上に複雑になる。作成済みのコマンド・パラメータ列や転送済みパラメータ列の認識，次のパラメータ列へのリンク，新しいパラメータ列の生成場所の定義などをCPUはすべて管理しなければならない。コントローラにそれらの情報を設定する

```
CHAR:   MOV   WORD PTR PITCHS, 0002H      ; 転送源アドレス・ピッチ転送：2
        MOV   WORD PTR DHH,    0017H      ; 水平方向ドット数転送：24
        MOV   WORD PTR DV,     0017H      ; 垂直方向ドット数転送：24
        MOV   WORD PTR EAD2H,  0000H      ; 転送源上位ワード絶対アドレス転送：0
        MOV   AX, ES:WORD PTR BUF2        ; 転送先 Y 座標初期値設定
        MOV   BX, ES:WORD PTR BUF3        ; 転送先 X 座標初期値設定
        MOV   DI, ES:WORD PTR BUF1        ; 色情報初期値設定
        MOV   DX, 0D000H                  ; 転送源下位ワード絶対アドレス初期値設定
        MOV   SI, ES:WORD PTR BUF8        ; コマンド・フラグ設定
CHAR_2: MOV   CX, 041AH                   ; ループ回数初期値設定：26 文字 4 行
CHAR_1: MOV   PLANES, DI                  ; 色情報転送
        MOV   EAD2L,  DX                  ; 転送源下位ワード絶対アドレス転送
        MOV   X,      BX                  ; 転送先 X 座標転送
        MOV   Y,      AX                  ; 転送先 Y 座標転送
        MOV   WORD PTR COM, SI            ; コマンド・フラグ転送
        ADD   DX, +30H                    ; 文字コード変更
        ADD   BX, +18H                    ; 転送先 X 座標変更
        INC   DI                          ;
        AND   DI, 0007H                   ; } 色情報変更
        DEC   CL                          ;
        JNZ   CHAR_1                      ; } 1 行 26 文字描画終了？
        SUB   AX, 0018H                   ; 転送先 Y 座標変更
        MOV   BX, 0000H                   ; 転送先 X 座標変更
        MOV   CL, 1AH                     ; 文字ループ回数初期値設定
        DEC   CH                          ;
        JNZ   CHAR_1                      ; } 4 行文字描画終了？
        DEC   WORD PTR BUF7               ;
        JNZ   CHAR_2                      ; } 26 文字 4 行描画×12 回終了？
        ;
```

図9　24×24ドット構成の文字1280字を展開するアセンブラ・プログラムの一部　CPUは80286である。コマンド・パラメータはこのようにムーブ命令（　　）だけで簡単に設定できる。

必要もある。また，コントローラ自身がシステム・バスの使用権授受のためのホールド制御をすることになるため，インタフェースが可能なCPUをコントローラが限定してしまうことになりかねない。

パラメータの生成時間と描画時間の関係が重要

パラメータ間接転送方式は，CPUの処理を止めることなく，できるかぎり先のパラメータ生成まで実行をねらっている。ところが実使用状態では，パラメータ直接転送方式と比べて利点はないと判断した。これらを比較するうえで，CPUによるパラメータ生成時間とコントローラの描画時間との関連が重要な因子となる。

塗りつぶし描画のように描画時間がパラメータ生成時間より遅い場合には，全体の描画時間は描画実行時間とほぼ等しくなる。しかし，間接転送方式では第1回目のデータ生成に要する時間だけ直接転送方式に比べ遅い（図10(a)）。

1ドットしか描画を実行しない極端な例を採れば，全体の描画時間はパラメータ生成時間に依存することになる（図10(b)）。ところが，間接転送方式ではパラメータ列を前もって生成しておくので，パラメータ生成処理に依存することなく描画を連続的に実行できる。しかし，これはあらかじめ作成しておいたデータ・ブロックを用いた描画に限定される。直接転送方式では，パラメータ生成時間に依存しつつ描画を実行していくので連続描画はできないが，全体の描画は常に短い時間で終了する。

つまり，間接転送方式では，ブロック内の描画時間は直接転送方式のようにパラメータ生成時間に依存しないので高速であるようにみえる。ところが，データ生成に費やす時間だけ実質的な全体の描画時間は長くなってしまう。

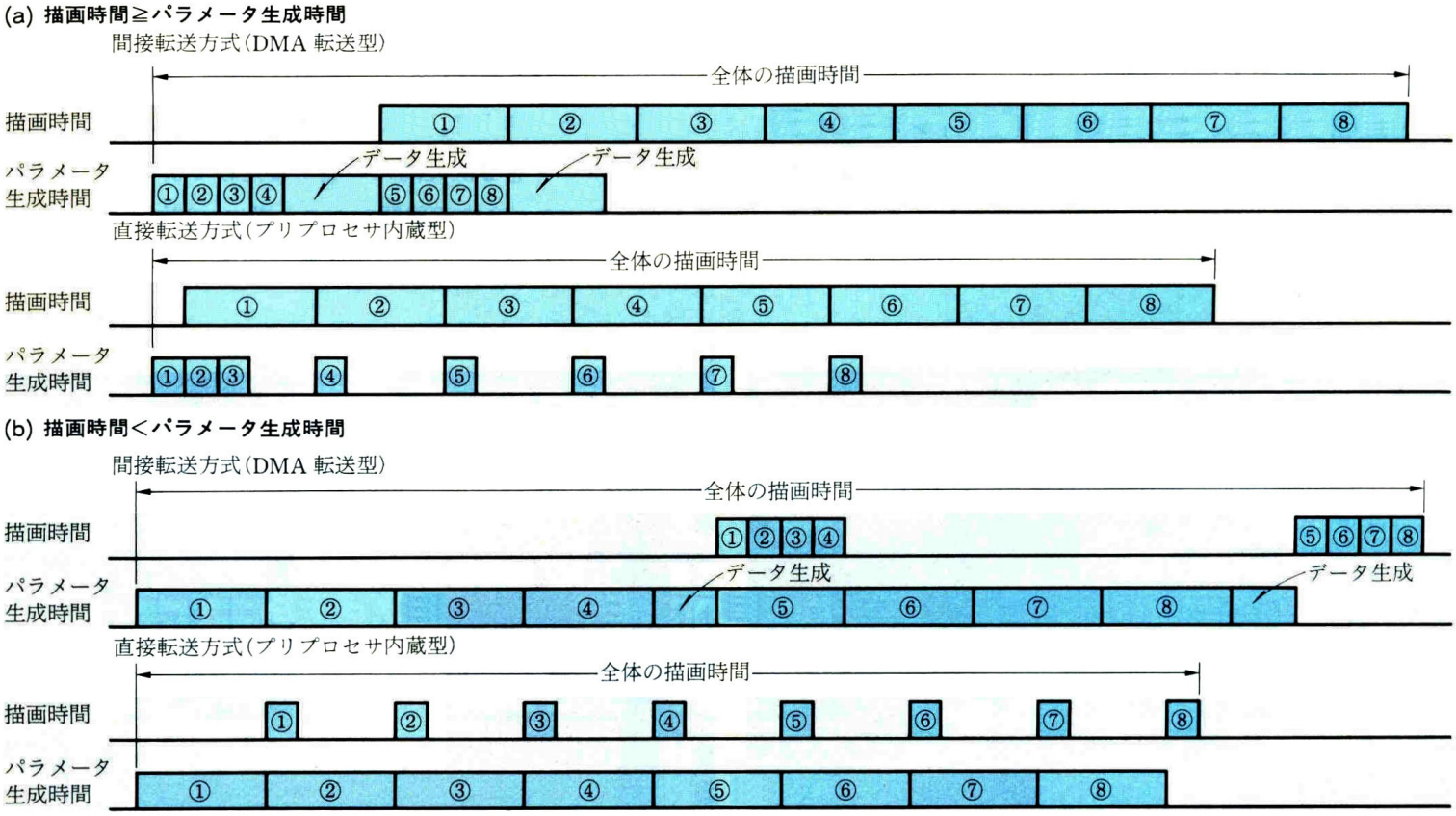


図10　実際の描画時間の比較　パラメータ直接転送方式と間接転送方式との間で，CPUとコントローラの動作を比較した。描画時間がパラメータ生成時間より長い場合(a)とその逆の場合(b)について全体の描画時間を評価している。パラメータ間接転送方式では，データ生成に要する時間だけ直接転送方式に比べ全体の描画時間は長くなる。

拡大・縮小，回転，塗りつぶしを高速処理する描画プロセサ

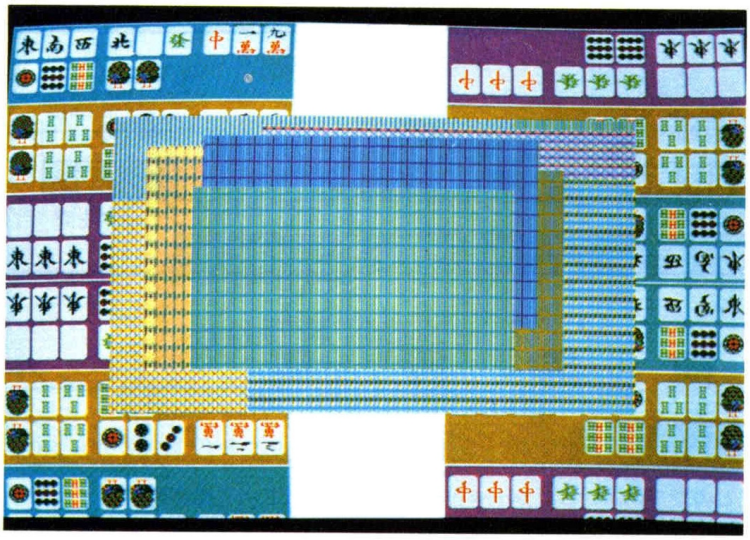
複数の処理結果を同一画面上に表示したり，異なる文書の内容を参照しながら新しい文書作成をするなどの目的からマルチウインドウ機能を提供している表示装置は一般的になってきた。表示メモリの内容を転送せずに，表示アドレスを変更するだけでマルチウインドウ表示を実現したコントローラも登場している[5]。これをハードウエア・ウインドウ機能と呼ぶ。

ハードウエア・ウインドウはデータ転送を一切しなくてもすむので高速である。しかし，ビット境界をもつウインドウを表示するには，表示データをいったんチップ内部に取り込む必要がある。高解像度の表示装置では，数百MHzにも達するドット周波数に対応した実時間制御をしなければならない。一般に，コントローラは，これより1桁低い周波数での動作しか保証できないため，表示画面の解像度に制約が生じてくる。また，表示可能なウインドウの枚数にも上限がある。高解像度になるほどウインドウ枚数を増加したいのだが，逆に，より少ない枚数に抑えられてしまう。水平帰線期間内に，その表示ラインで発生するすべての表示アドレスの変化位置と先頭表示アドレスを転送しなければならないからである。ウインドウ内のスクロールは高速になるが，これもまたウインドウ・メモリの容量に制約されてしまう。

クリッピングをウインドウ表示，ピッキングに利用

高速さという利点はあるが，装置設計段階でかなり大きな制約の生じるハードウエア・ウインドウ機能は搭載しないことにした。コピー機能によって表示データを転送しウインドウを形成する。表示内容が固定したポップアップ・メニューと変化するウインドウとでは異なる制御方法を採る。ウインドウ制御ではフレーム・バッファと同じメモリ構成のウインドウ・メモリを必要とするが，メニュー表示用のデータは2次元的にもつ必要はない。メニュー表示の必要になった時点で，1次元の連続したアドレス上に置いた原型データを2次元的に展開するだけでよい。このため，1次元データを2次元展開したり，逆に，2次元データを1次元データに変換することを可能にした。

(a)



(b)

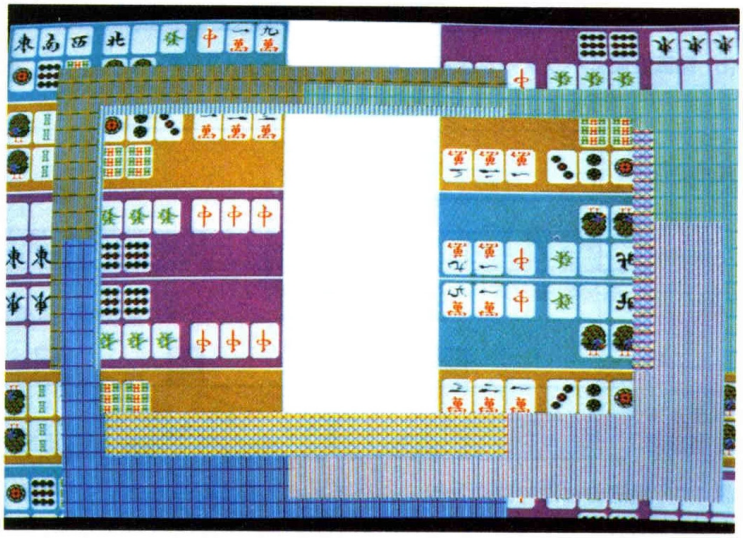


図11　クリッピング例　クリッピング領域を指定し，内側(a)，外側(b)を塗りつぶすモードを指定し，四辺形内フィルを実行した。

マルチウインドウを効率的に形成するために，さらにハードウエア・クリッピング機能を用意した。対角座標を2点指定することにより四辺形のクリッピング領域を定義する。その内部だけを描画するか，外側だけを描画するか，クリップ動作をしないかの動作設定ができる（**図11**）。表示最前部に位置するウインドウには有効に利用できる。ペイントのときには，このクリッピング領域の定義を境界点検索領域の外枠指定として使う。

クリッピングを用いると画面上に描画してある図形を特定し，抜き出すピック動作が可能になる。コントローラに描画をすべて任せてしまうと，CPUは描画位置をトレースしていない限り，ピッキングはできない。そこで，AGDCにこの機能をもたせた。マウスやタブレットによってクリッピング領域をあらかじめ目的の位置に移動させておき，2×2ドット程度の狭い領域を定義する。描画領域をクリッピング領域の外側に設定し，表示結果が変化しないような論理演算モードを選択して描画を再実行させる。クリッピングの発生したことを割り込みやフラグによってCPUは知ることができ，そのときに描画していた図形をピックできる。

プレーン型，ピクセル型の表示メモリを構成できる

AGDCの備える主な描画機能は表1に示した。しかし，そのすべての組み合わせが成立するわけではない（**表2**）。軸がX軸やY軸に平行ではない楕円の描画のように，AGDCが内蔵していない機能はCPUのソフトウエア処理によって補完しなければならない。このため，クリッピングのようなAGDCの実行する機能を他の描画で使用しているならば，CPUによる楕円描画にも適用できることが望ましい。この場合には，相対位置指定ができるドット描画を用いて，CPUで楕円を描画すればよい。表示メモリのうち，表示画面に対応するフレーム・バッファへの描画では，X-Y座標による位置指定ができる。漢字ROMやワーキング領域に対するデータ参照などには絶対アドレスによる位

表2　実際に使える描画機能の組み合わせ

		ピクセル構成	ハードウエア・クリッピング	拡大・縮小	線種の選択	塗りつぶしパターンの選択	転送先（描画先）位置指定	転送源（線種・塗りつぶしパターン）位置指定
ドット		○	○	×	○	——	座標	内蔵レジスタ
直線		○	○	拡大のみ	○	——	座標	内蔵レジスタ
四辺形		○	○	拡大のみ	○	——	座標	内蔵レジスタ
円・円弧		○	○	×	○	——	座標	内蔵レジスタ
楕円・楕円弧		○	○	×	○	——	座標	内蔵レジスタ
ペイント		×	○	——	——	○	座標	内蔵レジスタ・先頭アドレス
フィル	四辺形内	○	○	——	——	○	座標・絶対アドレス	内蔵レジスタ・先頭アドレス
	円内	×	○	——	——	○	座標	内蔵レジスタ・先頭アドレス
	台形内	×	○	——	——	○	座標	内蔵レジスタ・先頭アドレス
	三角形内	×	○	——	——	○	座標	内蔵レジスタ・先頭アドレス
コピー	通常	○	○	○	——	——	座標・絶対アドレス	座標・絶対アドレス
	90度回転	×	○	×	——	——	座標・絶対アドレス	座標・絶対アドレス
	傾斜	×	○	×	——	——	座標・絶対アドレス	座標・絶対アドレス
	任意角回転	×	○	○	——	——	座標・絶対アドレス	座標・絶対アドレス
プット・ゲット	通常	○	○	×	——	——	座標・絶対アドレス	——
	90度回転	×	○	×	——	——	——	座標・絶対アドレス
CPU直接描画		○	×	×	——	——	座標アドレス	任意

置指定を使える。

色情報を記憶するには次の2種の方法がある。

① カラー・プレーン型

②パックト・ピクセル型

プレーン型では，1ピクセルの色情報を色プレーンの大きさだけ離れたアドレスにまたがって格納している。したがって，直線描画のように1ピクセルごとに描画する場合は，1ドットずつすべてのプレーンに対する描画を順次実行しなければならない。ところがプレーン型は隣接するビット列を一括して処理したほうが高速になるコピー，塗りつぶし，回転，拡大や縮小処理に適している。パックト・ピクセル型は，表示メモリをアクセスする最小単位(1ワード)にピクセル情報をすべて詰め込んでいるので，1ピクセルを同時に描画できる。1ピクセルを単位として描画していく図形描画では，プレーン型よりも高速である。しかし，1ピクセルを構成するドット数を変更すると，描画処理の流れや描画マスク回路などのハードウエアが大きく変わるので拡張性に欠ける。

AGDCでは，プレーン型の表示メモリに対してはすべての描画機能を適用できる。しかし，ピクセル型の得意とする図形描画については，ピクセル型の表示メモリにも対応できるようにした。このプレーンとピクセルのモード選択は描画時に動的に切り替えることができるので，プレーン型とピクセル型の2種の構成をもつ表示メモリを実装し，それぞれの特徴を生かすことも可能である。

GDCとは異なり，プレーン型であっても一つの描画コマンドの発行によってカラーの描画を実行する。図形描画や塗りつぶしでは描画先にある複数プレーンに対して，最大2種類の論理演算を指定できる。たとえば，4面のメモリ・プレーンに対し，2面ごとに別々の論理演算を実行できることになる。コピーやプット/ゲットでは同様の処理を転送源および転送先に対して実行できる。

ドット・マトリクス型のカラー・プリンタにカラー画面のハードコピーを打ち出すことは容易である。複数のプレーンに格納してある転送源データ間で各インク・リボンに適応した論理演算を施し，90度回転したデータをシステム・バスに読み出す90度回転ゲット・コマンドを1回発行すればよい。これで縦方向16ドット1行分の印字ヘッド・ピンを直接駆動するドット列をAGDCから取り出すことができる。

16/N倍拡大，N/16倍縮小が可能

拡大・縮小コピー時の倍率は，拡大時$16/N$倍，縮小時$N/16$（Nは1から16の整数）とした。等倍近辺の倍率きざみを多く取れるので実用的である。拡大・縮小の倍率に制約をつけたのは，簡単なハードウエアによって高速な拡大・縮小コピーを実現するためである。たとえば15/16倍の縮小を考えてみよう。表示メモリから読み出した16ビットのうち，あらかじめ決められた1ビットを間引くだけでよい。縮小の場合，縮小率によってどのビットを間引くかはあらかじめ決定しておける。

縮小したデータはシフト，マスクなどの処理を通して再び表示メモリに書き込む。最終ワードの場合，拡大・縮小の終了していないビット数が16ビットあるとは限らない。このような処理では描画マスク演算をすることは難しい。AGDCでは，そのときの倍率と描画の終了していないビット数を基に，あらかじめマスクの形式を記入したテーブルを参照することによりマスクを生成している。

垂直方向の拡大・縮小は処理速度に敏感ではないので，ソフトウエアによる計数処理をしている。2倍なら単純に同じラインを2度描画するように拡大し，縮小ならば単純に間引く。文字や図形を扱うグラフィックスでは，1本の線が抜けたり，間隔がなくなると，視覚を通した情報量が激減してしまうことがある。罫線が消えてしまわないように直線描画するなど，縮小後のデータに対し応用に合わせたなんらかの補完処理をする必要があると思っている。

塗りつぶしパターンの参照方法として，次の3種類を用意した。

① 内蔵レジスタ参照

② 表示メモリ参照

(a)すべてのプレーンに共通なパターンを参照

(b)各プレーンに独立のパターンを参照

① はいわゆるベタ・パターンである。垂直方向やプレーン方向へは同じパターンの繰り返しになる。画面のクリア

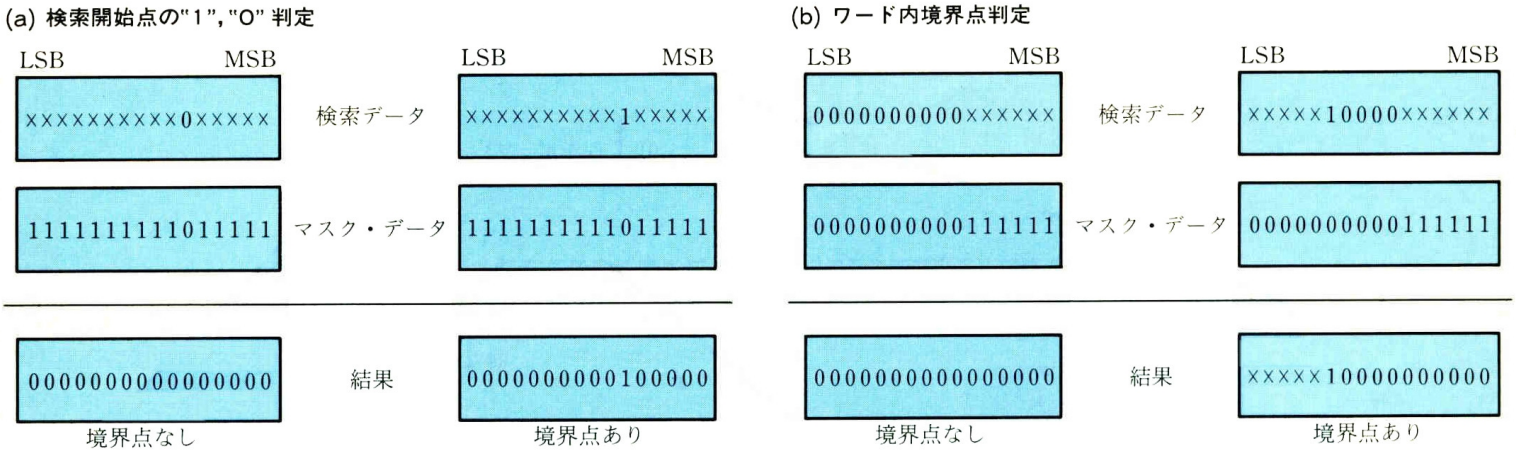


図12　境界点の判定　検索データにマスクをかけた結果に対し，そのワード内に境界点が存在するか，存在すればそのドット位置はどこかを知ることができる。図中の“×”は任意の値（“0”または“1”）を示す。

に使うことが多いだろう。②(b)の参照方法を用いると，水平，垂直，プレーン方向にすべて異なるパターンを参照できるので，1ドットごとに異なる色彩を着けた塗りつぶし（タイリング）が可能となる。塗りつぶすラインの Y 座標に依存して塗りつぶしパターンの参照位置の初期値算出や変更をする機能を内蔵している。

境界点検索を高速にすればペイントは速くなる

同じ大きさの領域を塗りつぶすならば，フィルのほうがペイントより速い。フィルを実行するための座標は高速に発生できる。一方，ペイントでは塗りつぶし領域を特定するために，表示メモリの内容を読み出し，境界点を検索する複雑な作業を伴うからである。ペイントの速度を上げるには，この境界点検索を短時間で終了させることが最優先課題となる。新規に開発した境界点検索器は16ビットの検索データを与えた直後に，次の情報を描画プロセサに返すことができる。

① 境界点が存在するかどうか

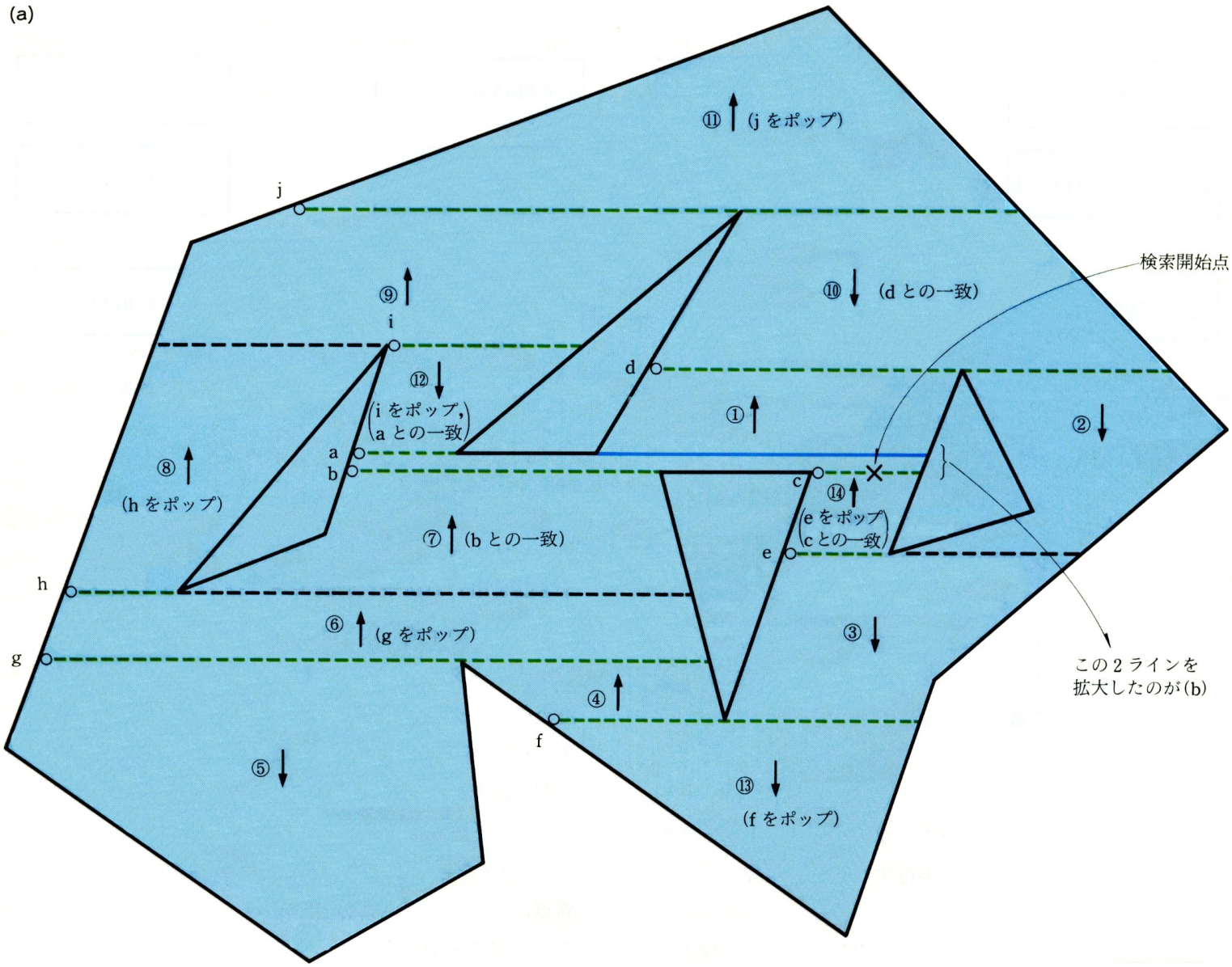
② 存在する場合には，そのドット位置

境界点がその検索データ内に存在していなければ，次の隣接したワードの内容を読み，そこから検索データを作る。カラー表示の場合には，境界点は境界色として与えられる。プレーンが3枚ならば，三つのプレーンのデータを連続してプレーン当たり2クロックのうちに読み取りながら，境界色指定にしたがってプレーン間でデータの論理演算を実行し，検索データを生成している。すべてのプレーンに対する読み出しが終了した時点で，16ピクセル(1ワード)の検索データの生成が終了する。

検索データに対しては，検索対象とする領域を特定するためにマスクをかけることができる(**図12**)。図12ではマスク・データが“0”のビットのみ検索対象としている。検索開始点そのものが境界に含まれているかどうかを判定するには，検索開始点のデータだけを取り出せるようにマスク・データを与える（同図(a)）。その結果が“0”ならば境界点はそのワード内に存在しない。ワード内のある1点から左右どちらかの方向への検索を続行していく場合には，検索が終了した部分を再び検索対象領域としてしまわないために，その点からワード境界までの間の検索データにマスクをかける（同図(b)）。ワード内に境界点が多数存在する場合には，同じ検索データに対してマスクを変えるだけですむ。境界点検索器の出力する位置データは塗りつぶし領域の大きさを算出するためだけでなく，そのワード内での検索マスク・データを生成するためにも使っている。

上下2ラインを読み取り領域のまわり込み点を抽出

閉領域の塗りつぶし順序と境界領域データのプッシュやポップの様子を**図13**(a)に示した。×印から検索を開始し，①～⑭の順に塗りつぶしていく例である。まわり込み



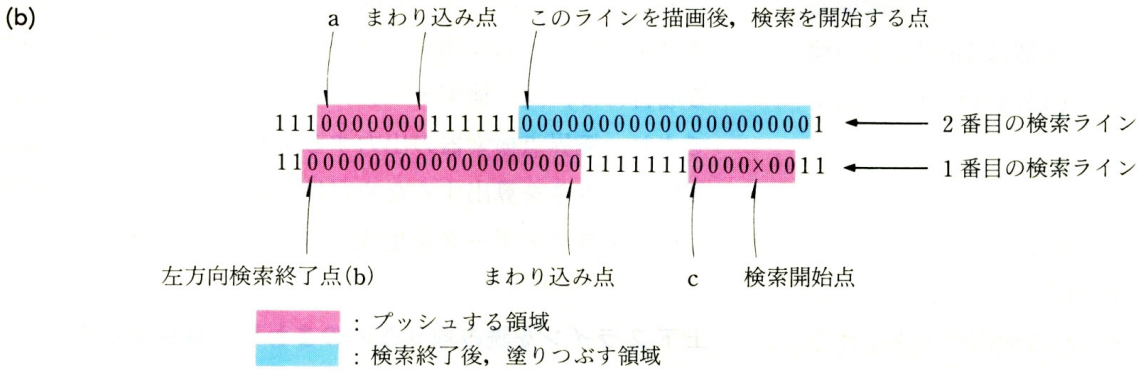


図 13　ペイントの動作手順　任意の閉領域は，複数の領域に分割して塗りつぶす(a)。図では，①～⑭までを，矢印の方向に塗りつぶしている。(a)の一部を拡大し，(b)に示した。境界点検索の手順を示している。境界点検索処理は“0”から“1”または“1”から“0”への変化点の検索を基にした閉領域の特定，まわり込み点の抽出，検索終了判定，閉領域情報のプッシュやポップ，実際の閉領域内塗りつぶしを含む。

点の発生により，閉領域を14個の領域に分割した。①などの横に付けた矢印は塗りつぶしを実行していく方向を示している。

境界点検索を含む塗りつぶしは次のように進む。図中に示した検索開始点から検索を始めるとしよう。検索開始点を含むラインと，その上のラインを同時に検索する。1ラインの検索途中で左右の境界点を見つけたときには，その部分の情報（閉領域情報）をスタックにプッシュする。この例では，a，b，cをプッシュすることになる。①では図中の━━から塗りつぶしを実行し，さらに上のラインの検索へ移る。①の検索の最後では，②へのまわり込み点が発生するのでdをプッシュし，②の塗りつぶしを始める。このように，塗りつぶしを実行しながら，閉領域情報を表示メモリ上に置いたスタックに積んでいく。

⑤の塗りつぶしが終了した時点ではa～gがスタックにプッシュされている。⑥の領域では，スタックをポップし，④の検索の終わりでプッシュしたgから検索，塗りつぶしを始める。また，⑥ではその後hをプッシュすることになる。

検索後の塗りつぶしを実行するときには常に，その検索で特定した閉領域情報とスタックの内容との一致を調べている。⑦の塗りつぶしでは，bとの一致がとれたので，bの閉領域情報にフラグを立て，ポップ後の検索を再びしないようにしている。

境界点検索では，上下2ライン分の表示メモリ内容を交互に読み取る方法を採った。この方法を使うと，塗りつぶし領域にあるまわり込み点の抽出，検索途中で領域が確定したときのスタック動作，および，検索終了の判定などの境界点検索処理の流れ[7]を明瞭にできる。2ライン分のデータを読み，それぞれの境界点検索結果を比較しながら境界点を検索していく。検索途中で塗りつぶし領域が確定できたときは，前述したようにあらかじめ定義しておいた表示メモリ上のスタック領域に位置や領域の大きさなどを含む6ワードの情報をプッシュする。

図13(b)に同図(a)の一部を拡大した。この図でもう少し詳しい動作を説明しよう。

まず，検索開始点そのものが境界に含まれていないことを確認し，その後“0”から“1”への変化点を検索する。その結果，点cが得られる。次にcより1ラインだけ上の点の境界点判定をする。その点が“1”なら左方向の境界が見つかったことになるので，右方向へ検索していく。この例では，“0”なのでまず左方向へ検索することになる。左方向検索を開始するときには1番目の検索ラインでは“1”から“0”，2番目の検索ラインでは“0”から“1”へ変化する点を検索する。

一般に，“0”から“1”への変化点が境界点となり，その上または下の点が“1”であれば検索終了点となる。“0”であればまわり込みの可能性がある。したがって，さらに“1”から“0”へ変化する点の検索をする。この処理を検索終了点が見つかるまで続ける。

左方向の検索終了点bを見つけ出すと，右方向への検索を始める。このとき，検索途中でaとbの閉領域情報はすでにプッシュしてある。右方向検索では，まずcの閉領域が確定するのでプッシュする。右方向検索が終了した時点で塗りつぶしを実行するラインが決まる。このラインの左端の点を検索開始点として①の残りの領域に対する検索を始める。

バッファ・レジスタを利用し90度回転などを実行

90度回転コピーを高速化するために16ワードのバッファ・レジスタを内蔵した。このバッファは，そのアドレス・バスとデータ・バス接続の制御によって，行方向から書き込んだデータを，あたかも列方向から読み出すこともできる構造をもっている。さらに，転送源と転送先の転送領域のビット位置の相対関係を判断して，バッファへのデータ転送に無駄が生じない流れ制御も採用した。このバッファ・レジスタは，ゲット/プット時には転送データを一時的に記憶するFIFOとしても機能する。

拡大・縮小を伴った任意角回転コピーは直線描画によって実行している。一般に，図形を回転させると，図形描画領域内に描画をしない点が発生する。軸に対する回転角度が45度になると，この現象は最も顕著になる。このような特異点を判別して，隣接する点への描画を，この特異点に対しても同様に実行するかを選択できる。

実際の実行速度を各種ベンチマーク・テストで評価

グラフィックス・コントローラの描画速度評価の方法は各社各様である。評価環境の条件も明確でないことが多い。したがって，単純にそれらの値から描画速度を比較することは難しい。コントローラの性能を「1ドット当たり何ナノ秒」とか「1秒で何ベクトル」と表現することがあるが，これはいわば最大瞬間風速的な値である。描画のために表示メモリを占有できる時間比率やコマンド・インタフェースなどのシステム的要因は考慮していない。実際の描画速度とは大幅に異なってくる場合も多い。

装置設計者がコントローラ採用の可否を判断する材料として有用な数値はシステムとして組み上げたときの描画速度であろう。そのためには，直線を1本だけ描画したり24×24ドットの文字を1文字だけ描画したときに要した時間を基に1秒間に描画できる直線の本数や文字数を算出する方法も良くない。描画を連続的に実行した場合に，描画と描画との間隔がどの程度あいてしまうのかについて評価していないことになるからである。

表3　描画サイクル数　この値だけを基にして，全体の描画速度を算出してしまうのは良くない。塗りつぶしでは参照パターンの違いによって，サイクル数が異なるが，塗りつぶし全体の描画速度はほとんど変わらない。

描画内容	描画サイクル数
直線	4クロック/ドット（ピクセル）
四辺形のうちの水平直線（プレーン構成）	4クロック/ワード
円，楕円	6クロック/ドット（ピクセル）
コピー	
データ置き換えのみ	2クロック/ワード
論理演算実行	4クロック/ワード
塗りつぶし	
内蔵レジスタ参照，データ置き換えのみ	2クロック/ワード
内蔵レジスタ参照，論理演算実行	4クロック/ワード
表示メモリ参照，論理演算実行	6クロック/ワード

われわれは，図形やグラフィックス文字描画の場合には，同一種類の描画を連続的に実行したときの総合的な描画速度を評価することにした。あらかじめ，座標値を与えておき，CPUが描画前処理を開始した時点から最後の描画が終了するまでの時間をロジック・アナライザを用いて実測した。パラメータ間接転送方式によるコントローラの場合には，DMA転送データやコマンド・リストの作成処理も含めた時間として表現することになる。CPUのデータ転送速度に依存してしまうゲットおよびプットは除外した。表示メモリとしてデュアル・ポート・メモリを使用しているが，そのリフレッシュのために6%が消費されたため，93.7%の描画可能タイミングが確保されている。

理論的には2〜6クロックで1ワードを処理

実測値の前に，描画実行に必要なクロック数(理論値)を**表3**に示す。直線はGDCと同一のクロック数で描画するが，円，楕円はむしろ遅い。これは，楕円の描画アルゴリズムを基に円を描画していること，一つのコマンドによって円全体を描画できるようにしたことによる。プレーン構成を採る表示メモリに対し，四辺形を描画するときには，水平線の部分は16ドットを1回の描画サイクルで描画できる。転送源と転送先をともにビット境界で指定できるコピーを，転送源または転送先に対する論理演算を実行しつつ，1ワード当たり6クロックで実行する。転送先の既存データを無視し，すべて転送源データで置き換える設定をすると，1ワードのコピーは4クロックで終了する。

塗りつぶしでは，内蔵レジスタに格納してある塗りつぶしパターンを参照する場合と，表示メモリ上のパターンを参照する場合とで実行サイクルが変わる。表示メモリ上のパターンを参照するには，読み出しサイクル2クロック分だけ，内部レジスタを参照した場合に比べて時間がかかる。四辺形内塗りつぶしでは，コピーの場合と同様に，データ

表4　描画時間の計測値　ロジック・アナライザを用い，最初に描画パラメータの発生を指示してから描画を終了するまでの実行時間を実測した。単位はms，クロックは8 MHzである。

(a) 図形描画の描画時間

	プレーン構成 1面	プレーン構成 3面	ピクセル構成 4ビット	描画内容
直線	37.8	113.4	37.8	(319, 239) を始点とし，(0, 0)から終点座標をX方向±10しながら，240ピクセルの直線総数128本，Y方向±10しながら，320ピクセルの直線を総数96本。
四辺形	15.6	46.8	37.8	(0, 0)-(639, 479) から (235, 235)-(404, 244) まで対角座標を±5しながら描画した四辺形を総数48個。
円	30.0	69.0	30.0	中心 (319, 239) 固定，半径239から半径を−5しながら描画した同心円を総数46個。
楕円1	34.8	80.6	34.8	中心 (319, 239) 固定，X方向半径239から半径を−5しながら描画した同心楕円を総数46個。X方向半径とY方向半径の比率は4：3。
楕円2	45.0	104.2	49.8	楕円1の条件のうち，X方向半径とY方向半径の比率を3：4とした同心楕円。

(b) 塗りつぶし描画の描画時間

	ベタパターン	タイリング1	タイリング2	描画内容
画面クリア	28.8	——	——	640×480ドット×3面×2組（11万5200ワード）に対するクリア。
四辺形内フィル	7.1	13.5	13.8	400×300の矩形領域に対する論理演算付き塗りつぶし。
円内フィル	9.3	9.3	9.3	半径150ドットの円形領域に対する論理演算付き塗りつぶし。
楕円内フィル1	6.7	6.7	6.7	X方向半径150ドット，X方向半径とY方向半径の比率は4：3の楕円形領域に対する論理演算付き塗りつぶし。
楕円内フィル2	12.3	12.3	12.3	X方向半径150ドット，X方向半径とY方向半径の比率は3：4の楕円形領域に対する論理演算付き塗りつぶし。
三角形内フィル	3.0	3.0	3.0	(152, 419)-(320, 240)-(459, 320) の3点で囲まれる三角形領域に対する論理演算付き塗りつぶし。
台形内フィル	5.2	5.2	5.2	(0, 150)-(300, 150)-(30, 0)-(270, 0) の4点で囲まれる台形領域に対する論理演算付き塗りつぶし。
ペイント	94.5	96.7	97.8	中心 (320, 240) で半径100と220の円，(0, 0)-(639, 479) の四辺形，8個の三角形で形成する3個の閉領域に対する任意閉領域内塗りつぶし。

(c) コピーの描画時間

	1面→1面	3面→1面	1面→3面	3面→3面	描画内容
通常コピー1	12.0	29.2	24.3	36.3	640×480ドットの矩形領域の論理演算なしでの転送。
通常コピー2	18.0	29.2	40.5	52.1	640×480ドットの矩形領域の論理演算付きでの転送。
90度回転コピー	20.2	29.2	43.1	61.1	480×640ドットの矩形領域の論理演算付きでの90度回転をともなった転送。
拡大コピー	34.1	——	56.6	102.3	16/13倍に拡大したデータを640×480ドットの転送先矩形領域に対して，論理演算付きで転送。
縮小コピー	63.3	——	95.6	153.0	640×480ドットのデータを13/16倍に縮小して論理演算付きで転送。
任意角回転 拡大・縮小コピー	35.2	——	71.6	103.1	200×200ドットのデータを14/16倍に縮小し，X，Y軸に対し$\pi/8$だけ回転して論理演算付きで転送。

	1面→1面	1面→3面	描画内容
文字フォント通常展開	91.5	155.2	24×24ドット構成のフォント1248字を通常展開。
文字フォント傾斜展開	119.2	196.5	24×24ドット構成のフォント1248字を傾斜展開。

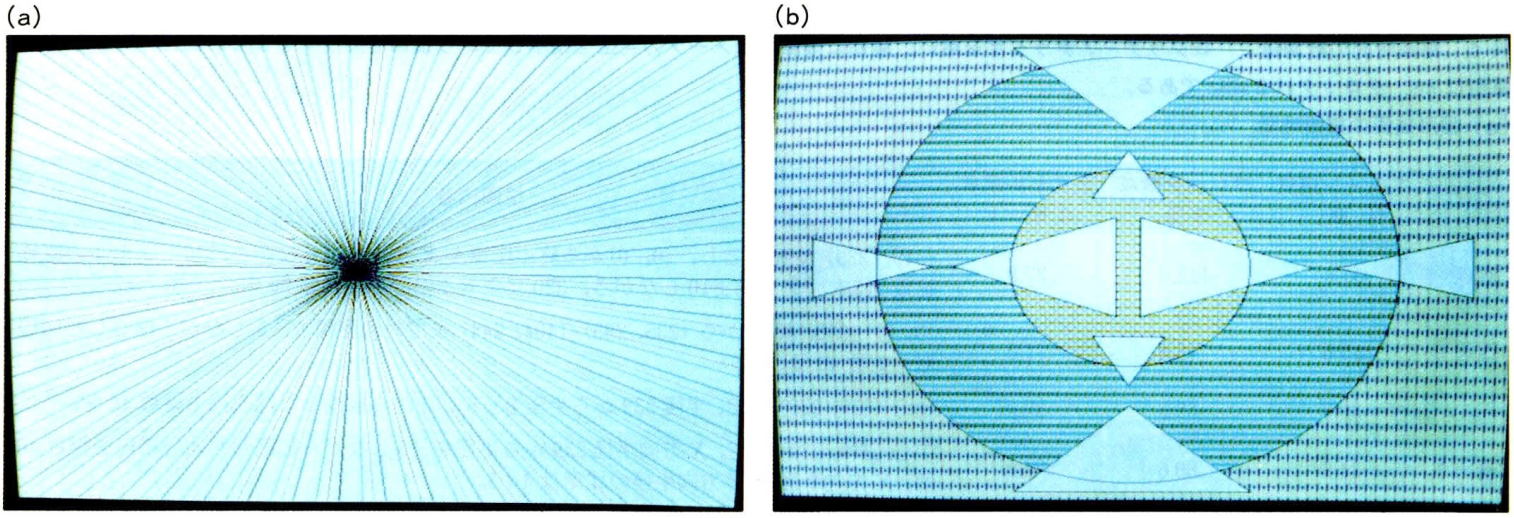


図14　描画時間の計測に用いたベンチマークの処理例　(a)は直線(表4(a))，(b)はペイント(表4(b))の描画例である。

の置き換えだけの設定をすると2クロックで16ビットを塗りつぶせる。

任意閉領域内の塗りつぶしは特に高速

図形の描画，塗りつぶし，コピーの実行時間を**表4**の(a)，(b)，(c)にそれぞれ示した。

1面の表示メモリ・プレーンのみに対する描画時間と3面（8色表示）に対する描画時間を比べると，直線（**図14**(a)）の場合には3倍になっている。しかし，円，楕円では2倍強にしかならない(表4(a))。円，楕円の描画に必要な6クロックのうち，1面に対する描画で4クロックかかる描画実行期間のみ3倍の12クロックになるからである。

表4(b)に示した画面クリアは塗りつぶしの一例である。内蔵レジスタに格納している塗りつぶしパターンをクリア用のパターンとして参照しながら四辺形を塗りつぶす。四辺形の左辺と右辺がビット境界の場合は，そのワードの塗りつぶしにマスク処理が必要になり，リード・モディファイ・ライトを実行する。しかし，この画面クリアではビット境界はない。11万5200ワードを28.8 msで実行する。これから，1ワードを2クロックでクリアできることがわかる。四辺形内塗りつぶし以外の塗りつぶしでは，塗りつぶし速度は塗りつぶしパターンの参照方法にほとんど依存していない。塗りつぶしパターンの参照方法により読み出し時間は異なるが，これよりも塗りつぶし領域を確定するための座標計算や境界点検索に要する時間が長いからである。

任意閉領域内塗りつぶしは非常に高速である（表4(b)）。この例では図14(b)に示したように画面の約80％を塗りつぶしている。しかし，3面で構成する画面全体を他の3面へコピーする時間と比べて約2倍弱でしかない。コピーでは定められた単純な描画アドレス計算をすればよいが，ペイントでは境界点検索という複雑な処理をしている。この速度は，境界点検索を高速実行する前述したハードウエアと検索アルゴリズムの成果である。また，90度回転コピーと通常のコピーとの間で描画実行時間にはそれほど差異は生じてこない。通常のコピーの約2倍の時間で拡大，約3倍の時間で縮小ができると思ってよい。

24×24ドットの文字フォントを展開する描画時間は，大きな領域に対するコピーとは別個に評価した。ここではワード境界ではなくビット境界にフォントを描画している。描画開始座標を直線発生器により算出している傾斜展開と通常展開で実行時間にほとんど差は出ていない。並列処理とパイプライン処理が効率良く行なわれている証拠である。この連続的なグラフィックス文字描画の評価結果から計算して，1秒間に白黒で1万3640字，8色で8040字の24×24ドット構成のグラフィックス文字描画ができることになる。この値は描画サイクル時のクロック数を基にして机上で算出したものと比べると，かなり低い数字になるが，より実際的な数字である。

CGI準拠や高速化が可能

VDIあるいはその簡略版ともいえるCGIといったグラフィックス・インタフェース規格のとりまとめ作業が進んでいる。グラフィックス・コントローラが新しく開発されるたびに，これらの規格へどのように対応しているかが話題になるようである。結論から言えば，CGIの定めたコマンドを直接解釈できるコントローラが出現するのはまだまだ先のことになる。規格が完全に固まり，グラフィックス標準として一般に浸透し，実績のある枯れたソフトウエアになった段階で，ようやくLSI化に向かうことになろう。

「CGIに準拠している」とするコントローラもあるが，これには議論の余地がある。CGIインタプリタは，コマンド解釈部分とデバイス・ドライバ部分に大別できる。コマンド解釈部分はコントローラに依存しないため，異種のコントローラであってもCPUが同じであれば流用できる。CPUが異なっていてもC言語などで記述していれば問題はない。使用するコントローラによって差が出てくるのはデバイス・ドライバ部分である。われわれは，このデバイス・ドライバがいかに簡単に作成できるかによって，「CGIに対する準拠度」が決まると考えている。

AGDCは，CGIの備える描画機能をほとんど内蔵している(**表5**)。したがって，AGDCの必要とするパラメータを生成するだけでよい。複雑な描画アドレス生成や表示メモリを制御するソフトウエアの開発は要らない。内蔵していない描画機能を補完する場合だけ，ソフトウエア作成工数を割り当てればすむ。

3次元表示やイメージ処理へ向けさらに高機能に

AGDCはGDCの開発によって築いた設計基盤にほとんど依存しなかった。集積度が現在に比べると比較にならないぐらい低かったころに設計したGDCは，ハード・ワイヤド・ロジックによって限定された描画機能を達成しなければならなかったからである。まったく新しいシステム構想の下で，描画アルゴリズム，描画高速化のための専用ハードウエア，それらを効果的に制御するプリプロセサや描画プロセサ，個々のハードウエアやソフトウエアなど，すべてを設計基盤がない状態から徐々に積み上げながら開発していかなければならなかった。暗中模索の状態が長く続いた。AGDCはまったく新しい第2世代のグラフィックス・コントローラとして開発したことになる。そのため，どのようなコントローラが最適であるかについて，装置設計者の側に立って隅々にまで細かい配慮をした。

最高動作周波数は8MHzであるが，今後チップ寸法を縮小することによって高速化する予定である。ある機能が達成されれば，次により高度な機能をもつ製品を開発することになるのは当然の流れである。3次元グラフィックス機能への飛躍やイメージ処理への展開をすることになるだろう。さらに，デュアル・ポート・メモリが搭載する機能を活用したり，機能を新たに搭載することにより，効果的な機能分担をしていくことも考えられる。いずれにせよ，次期開発製品からはこのAGDCを基盤として，機能を追加していくことになる。今回経験したほどの障壁にはぶつからないですむであろう。

参考文献

1) 小口，「1ドットを800 nsで描画できるラスタ走査型CRT用グラフィック・コントローラLSI」，『日経エレクトロニクス』，1981年10月12日号，no. 275，pp. 186-209.
2) Pinkham, R., Novak, M. and Guttag, K., "Video RAM Excels at Fast Graphics," *Electronic Design*, Aug. 18, 1983, pp. 161-171.
3) 小林，「間断のないシリアル出力を可能にしたフレーム・バッファ用256Kビット・デュアル・ポート・メモリを開発」，『日経エレクトロニクス』，1985年8月12日号，no. 375，pp. 211-240.
4) Wientjes, B., Guttag, K. and Roskell, D., "First Graphics Processor Takes Complex Orders to Run Bit-mapped Displays," *Electronic Design*, Jan. 23, 1986, pp. 73-81.
5) 御法川，上野，吉田，武田，前島，桂，「座標で描画位置を指定でき，塗りつぶしやコピーなど豊富なコマンドを持つCRTコントローラ」，『日経エレクトロニクス』，1984年5月21日号，no. 343，pp. 221-254.
6) 稲葉，「ワークステーションのマルチウインドウ表示方法を比較する」，同上，1985年7月29日号，no. 374，pp. 141-161.
7) 小口，南野，樋口，「グラフィックス・ディスプレイ・コントローラ」，『トランジスタ技術』，1983年1月号，pp. 320-343. ●

次ページに表5，略語一覧を掲載。

表5　AGDCのコマンド一覧　Aではじまるコマンドは絶対位置指定，Rは相対位置指定を示す。コマンド名の最後につけたAは絶対アドレス指定，Cは座標指定，Dは描画プロセサの座標をもとにする描画，Mは座標を新しく設定し直す描画を意味している。

コマンド名	設定するパラメータおよび動作内容
READ_DP	描画プロセサのXとY座標をプリプロセサに転送
READ_COL	指定した座標の色情報をプリプロセサに転送
DOT_D	(X#, Y#)に1ドット描画
A_DOT_M	(X, Y)に1ドット描画
R_DOT_M	(X+DX, Y+DY)に1ドット描画
A_LINE_M	(X, Y)-(XE, YE)間の直線描画
A_LINE_D	(X#, Y#)-(XE, YE)間の直線描画
R_LINE_M	(X, Y)-(X+DX, Y+DY)間の直線描画
R_LINE_D	(X#, Y#)-(X+DX, Y+DY)間の直線描画
A_REC	(X, Y)-(XS, YS)で定義する四辺形描画
R_REC	(X, Y)-(X+DX, Y+DY)で定義する四辺形描画
CRL	中心(XC, YC)，半径DXで定義する円描画
ARC	中心(XC, YC)，半径DX，開始点(XS, YS)，終了点(XE, YE)で定義する円弧描画
CSEC	中心(XC, YC)，半径DX，開始点(XS, YS)，終了点(XE, YE)で定義する円弧扇形描画
CSEG	中心(XC, YC)，半径DX，開始点(XS, YS)，終了点(XE, YE)で定義する円弧弦形描画
ELPS	中心(XC, YC)，Y方向半径DY，XとYの方向半径2乗比DH, DVで定義する楕円描画
EARC	中心(XC, YC)，X方向半径DX，Y方向半径DY，XとYの方向半径2乗比DH, DVおよび開始点(XS, YS)，終了点(XE, YE)で定義する楕円弧描画
ESEC	中心(XC, YC)，X方向半径DX，Y方向半径DY，XとYの方向半径2乗比DH, DVおよび開始点(XS, YS)，終了点(XE, YE)で定義する楕円弧扇形描画
ESEG	中心(XC, YC)，X方向半径DX，Y方向半径DY，XとYの方向半径2乗比DH, DVおよび開始点(XS, YS)，終了点(XE, YE)で定義する楕円弧弦形描画
PAINT	境界点検索開始点(X, Y)，境界色指定DXによる任意閉領域内塗りつぶし 境界点検索開始点(X, Y)の色以外を境界色とする任意閉領域内塗りつぶし
A_REC_FILL_A	開始絶対アドレスEAD1，水平方向ドット数DH，垂直方向ドット数DVで定義する四辺形内塗りつぶし
A_REC_FILL_C	(X, Y)-(XS, YS)で定義する四辺形内塗りつぶし
R_REC_FILL	(X, Y)-(X+DX, Y+DY)で定義する四辺形内塗りつぶし
CRL_FILL	中心(XC, YC)，半径DXで定義する円内塗りつぶし
ELPS_FILL	中心(XC, YC)，Y方向半径DY，XとYの方向半径2乗比DH, DVで定義する楕円内塗りつぶし
A_TRI_FILL	(X, Y)-(XS, YS)-(XC, YC)で定義する三角形内塗りつぶし
A_TRA_FILL	(X, Y)-(XS, Y)-(YS, YE)-(XE, YE)で定義する台形内塗りつぶし
R_TRA_FILL	上辺左の点(X, Y)，右の点(XS, Y)，上辺と下辺の相対変位DY，第3点のXからの相対変位DX，第4点のXSからの相対変位XCで定義される台形内塗りつぶし
A_COPY_AA	転送源の転送開始ワードアドレスEAD2，ドットアドレスdAD2，転送先の転送開始ワードアドレスEAD1，ドットアドレスdAD1，水平方向ドット数DH，垂直方向ドット数DVで定義する領域間の転送
A_COPY_AC	転送源の転送開始ワードアドレスEAD2，ドットアドレスdAD2，転送先の転送開始点(X, Y)，水平方向ドット数DH，垂直方向ドット数DVで定義する領域間の転送
A_COPY_CA	転送源の転送開始点(XS, YS)，転送先の転送開始ワードアドレスEAD1，ドットアドレスdAD1，水平方向ドット数DH，垂直方向ドット数DVで定義する領域間の転送
A_COPY_CC	転送源の転送開始点(XS, YS)，転送先の転送開始点(X, Y)，水平方向ドット数DH，垂直方向ドット数DVで定義する領域間の転送
R_COPY_CC	転送源の転送開始点(XS, YS)，転送先の転送開始点(XS+XC, YS+YC)，水平方向ドット数DH，垂直方向ドット数DVで定義する領域間の転送
PUT_A	転送先の転送開始ワードアドレスEAD1，ドットアドレスdAD1，水平方向ドット数DH，垂直方向ドット数DVで定義する表示メモリ領域への主記憶からの転送
PUT_C	転送先の転送開始点(X, Y)，水平方向ドット数DH，垂直方向ドット数DVで定義する表示メモリ領域への主記憶からの転送
GET_A	転送源の転送開始ワードアドレスEAD1，ドットアドレスdAD1，水平方向ドット数DH，垂直方向ドット数DVで定義する表示メモリ領域から主記憶への転送
GET_C	転送源の転送開始点(X, Y)，水平方向ドット数DH，垂直方向ドット数DVで定義する表示メモリ領域から主記憶への転送

略語一覧(ABC順)

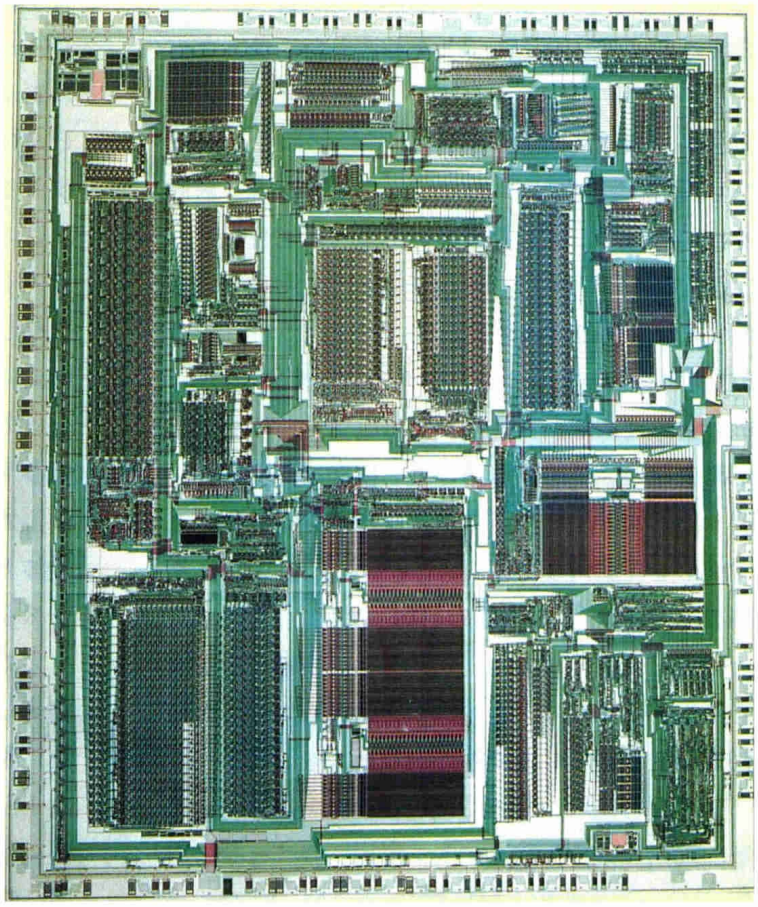
AGDC : Advanced Graphics Display Controller
bitblt : bit block transfer
CAS : column address strobe
CGI : computer graphics interface
CPU : central processing unit
FIFO : first-in first-out
GDC : Graphics Display Controller
ISSCC : International Solid State Circuits Conference
JIS : Japan Industrial Standard
LSI : large scale integrated circuit
OA : office automation
RAM : random access memory
RAS : row address strobe
ROM : read only memory
VDI : virtual device interface

Article: A Graphics Display Controller LSI (Successor of μPD7220) with Enhanced Copy and Paint Functions

Tetsuji Oguchi, Mitsuro Ouchi,
Ryuji Horiguchi, Toshikazu Chiba ... NEC Microcomputer Product Division

Keishi Uno ..... NEC 1st LSI Division

NEC has developed a new graphics display controller LSI -- AGDC (μPD72120). Based on its predecessor GDC (μPD7220), which was placed on the market in 1982, the features of μPD72120 include an enhanced bit map processing function, a copy function of rectangular areas which include enlargement, shrinkage and rotation and a fill function of triangles and trapezoids. Within the LSI, a preprocessor for executing draw preprocessing is mounted in addition to a drawing processor. The CPU, preprocessor and drawing processor are arranged for 3-stage pipeline processing to enhance system performance. Capacity of the display memory can be increased to 32 M bytes (max.) using dual port memory chips. μPD72120 operates with clocks up to 8 MHz. (NIKKEI)



The article was translated from the Japanese in the "Nikkei Electronics / Feb. 23 , 1987".

A Graphics Display Controller LSI (Successor of μPD7220) with Enhanced Copy and Paint Functions

NEC announced the graphics draw/display LSI μPD7220 (GDC) at the ISSCC in February of 1981. Since then, this LSI has been used in a wide area of applications, such as OA equipment, fish-finder, photocomposers and satellite terminals. The GDC can directly control a display memory of up to 512 K bytes which is separated from the system bus. This LSI is equipped with functions to draw straight lines, arcs, graphics characters, etc. in high speeds. (Ref. 1) Although speeds of drawing one dimensional objects such as straight lines and arcs are thus increased, the GDC is not equipped with a two dimensional draw function (i.e. that of areas), such as paint and character font expansion for the bit map memory. The CPU then needs to perform their supplementary functions.

The newly developed μPD72120 (AGDC) aims to minimize supplementation by the CPU. Rich draw functions are added to the graphics controller and the drawing speeds are increased. As far as graphic draw functions are concerned, differences between the GDC and AGDC are not remarkable except that an ellipse, an arc, a sector and a chord can now be drawn by using single commands.

During the design of the AGDC, it was attempted to mount as many bit map processing functions (which were missing in the GDC) as possible: data transfer between display memories (copy or bitblt function), paint function in an arbitrary closed area and unconditional fill functions in a circle, an ellipse, a triangle and a trapezoid. (See Figure 1 (a).) Among these functions, the copy function plays an important role in expanding character fonts and displaying the window. Also, during a copy, a graphic can be slanted or rotated through an arbitrary angle and, under some restrictions, enlarged or shrinked by an arbitrary factor. (See Figure 1 (b).) Together with the enhanced bit map functions, a put function and a get function, which are used when one dimensional data stored on the main storage are expanded in a two dimensional manner on the display memory or vice versa, are also added.

Although these functions can also be offered using a general processor, it is then not practical to expect sufficient drawing speeds. (Ref. 4) Typical drawing speeds of the AGDC, when it is operated at a clock frequency of 8 MHz are given below. While drawing a straight line, the AGDC takes 500 ns to draw a bit or 1 pixel. If a copy function is used, 13,640 characters of 24 x 24 dot character font can be expanded in one second.

Rapid Shift From Character Display To Graphics Display:

As the memory price has lowered these years, there have been rapid changes in the display technology, that is, from character display to graphics display. Even for word processors, it now seems that handling of graphics data is a minimum condition. But graphics display requires a large memory capacity and its control is not easy. To change a character on a character display screen, for example, it is only necessary to rewrite one byte or one word of the character code memory contents. On a graphics display screen, however, it becomes necessary to expand the character font on the display memory again. If this is processed by the CPU, the rewriting speed is too slow. And painting a complex figure is too great a burden for the CPU.

For this reason, there have been attempts to implement high-speed bit map functions using special gate array circuits. But, due to restraints on the integration level and speed of gate arrays, it has not been possible to execute macro drawing functions like the filling of a triangle. So, at each of these small steps, the CPU needed to participate in the drawing. If the drawing speed is evaluated in small-step units, the speed of the array circuit is high. But from the viewpoint of an overall system, its performance often fails to meet expectations.

From this background, since the GDC appeared, many graphics controller LSI's with enhanced drawing functions have been announced. Table 1 lists some typical controller LSI's. These products show different features depending on design objectives of the manufacturers. Am95C60 (of Advanced Micro Devices, Inc., U.S.A.) and AGDC aim to implement high-speed drawing functions by means of hardware, while HD63484 (of Hitachi) tries to emphasize its display control function. 82786 (of Intel, U.S.A.) mainly tries to enrich its display control function and TMS34010 (of Texas Instruments Inc, U.S.A.) is equipped with a built-in processor so that drawing software can be created freely.

Another important move to be noted concerning graphics display is the appearance of dual port memory chips with a line buffer. The first product was a 64 K dynamic RAM chip having a serial memory type built-in line buffer. (Ref. 2) After this, 256 K dual port memories with a random access type line buffer which are equipped with real-time data transfer function, pointer control function and write per bit function were announced.

The use of dual port memory chips markedly increases possible draw timings, to make the best use of the potential power of a continuous drawing controller.

Improve a System Performance

The AGDC is equipped with dual port memories. The AGDC has been designed to improve the system performance through the combined graphics control LSI and the CPU. Its internal architecture adopts a multi-processor configuration. Separate from the drawing processor, a processor especially for pre-drawn processings (called a preprocessor) is included.

The preprocessor performs pre-drawn processings such as address conversion from X-Y coordinates to absolute addresses. As the command interface, the CPU directly writes command parameters into the built-in register of the preprocessor. The preprocessor operates independently of the drawing processor and, as a result, the CPU can execute its pre-drawn processings such as command interpretation of CGI and BASIC and driving of the AGDC; the preprocessor can perform its pre-drawn

processings and the drawing processor can draw in a 3-stage pipeline manner when viewed as a system.

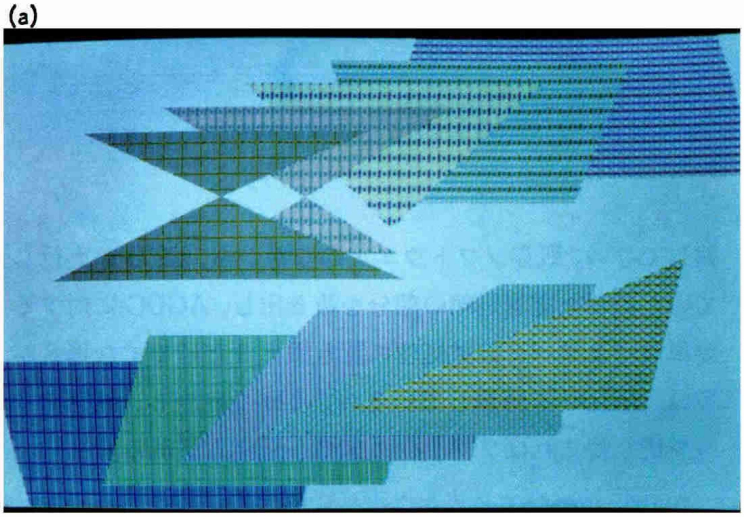
In addition, during the research of the drawing algorithm, a special hardware suiting the algorithm was designed to improve the drawing speed. The drawing processor was designed to tightly control these hardware components and, even when drawing types change, adapt to the various applications. Note that, in each step to be executed in a clock cycle of 125 ns (at 8MHz), up to 6 commands and direct branches can be executed. The execution of each command can use a command prefetch function as pipelining and, so, the operation of the processor itself includes both modes of parallel processing and pipeline processing.

Non-Compatible with GDC:

We greatly improved the functions rather than maintain compatibility with the GDC at the command and function levels. To maintain compatibility, it would have been necessary to use the GDC as the master, as was done in the past, and add an AGDC to improve the drawing functions. In this case, the existing software can be used but the functions of the AGDC are not available. However, it is not difficult to add routines which make use of the AGDC to the existing software. In many cases, in fact, it is simply enough to extract the complex drawing parts (executed by the CPU in the past) from among the existing software and replace them with simple parameter transfer processings for the AGDC. By judging whether or not an AGDC is mounted and, when mounted, switching the control flow, unification of the entire software is also possible.

We hope that the AGDC will be used in every application field which requires graphics functions, such as personal computers, word processors, work stations and laser printers.

(a)



(b)

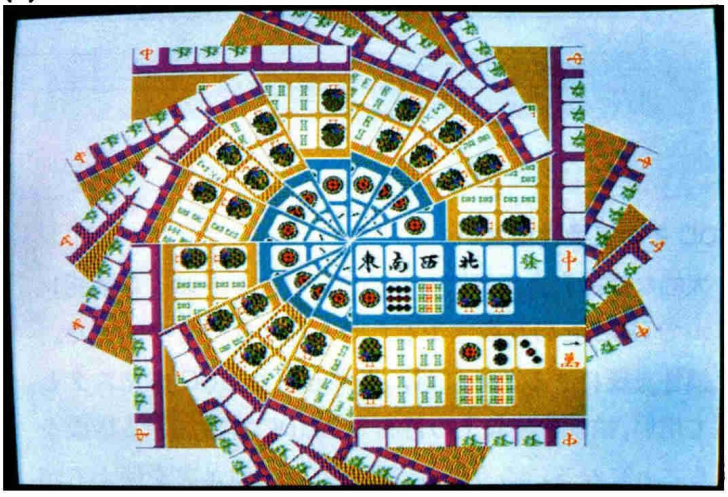


Figure 1

Drawings Using 640 x 480 Dot Display Screens

(a) A single command with its parameters as coordinates of apexes has painted the trapezoids.

(b) The result of a rotational copy of a rectangle. A reduce processing is executed as the rotation angle increases. This is because if the rectangle is rotated with a fixed magnification factor, the length of each side changes.

Table 1 Functional Comparison of Typical Graphics Controllers

	HD63484 (Hitachi)	Am95C60 (Advanced Micro Devices, Inc.)	TMS34010 (Texas Instruments Inc.)	82786 (Intel Corp.)	μPD72120 (NEC)
o Drawing function					
Straight line, rectangle	◎	○	○	◎	◎
Circle, arc	◎	◎	-	◎	◎
Ellipse, elliptic arc	◎	x	-	x	◎
Sector, chord	x	x	-	x	◎
Arbitrary closed area painting	△	◎	-	x	◎
Rectangle fill	◎	◎	-	x	◎
Circle/ellipse fill	x	◎	-	x	◎
Triangle fill	x	◎	-	x	◎
Trapezoid fill	x	x	-	x	◎
o Display memory data transfer					
Normal copy	◎	◎	◎	◎	◎
90 deg. rotation copy	◎	◎	-	◎	◎
Slant copy	x	x	-	x	◎
Enlarged/shrinked copy	x	○	-	x	◎
Enlarged/shrinked copy including arbitrary angle rotation	x	x	-	x	◎
o Hardware clipping	◎	◎	x	◎	◎
o Put, get	x	◎	-	x	◎
o Drawing position designation	X-Y coordinates	X-Y coordinates	X-Y coordinates	X-Y coordinates, absolute addresses	X-Y coordinates, absolute addresses
o Display memory configuration (maximum capacity)	Pixel type (2 M bytes)	Plane type (8 M bytes)	Pixel type (128 M bytes)	Pixel type (4 M bytes)	Plane type, pixel type (32 M bytes)
o Mapping of display memory into CPU	x	x	x	◎	◎
o Dual port memory control	x	◎	◎	◎	◎
o Separation of drawing from pre-processing	x	x	x	x	◎
o Separation of clock between display and drawing	x	◎	x	x	◎
o Display functions					
Screen partition	◎	x	x	◎	x
Enlarged display	◎	x	x	◎	x
Hardware window	△	△	x	◎	x
Cursor output	◎	x	x	◎	◎

In the table below,

△ : shows that the function is available but not sufficiently practical.
o : shows that only a part of the function is available.
- : shows that the function can be implemented by software.

Internal Architecture with A Multiprocessor Configuration

The AGDC is internally divided into the following five functional blocks. (See Figure 2.)

1 Preprocessor
2 Drawing processor
3 Display processor
4 Synchronization signal generator
5 CPU interface

The important feature of this architecture is its multiprocessor configuration consisting of three types of processors for drawing, displaying and preprocessing. And this configuration enables pipelining by the CPU, preprocessor and drawing processor. The drawing processor is equipped with a special hardware to improve its drawing speed.

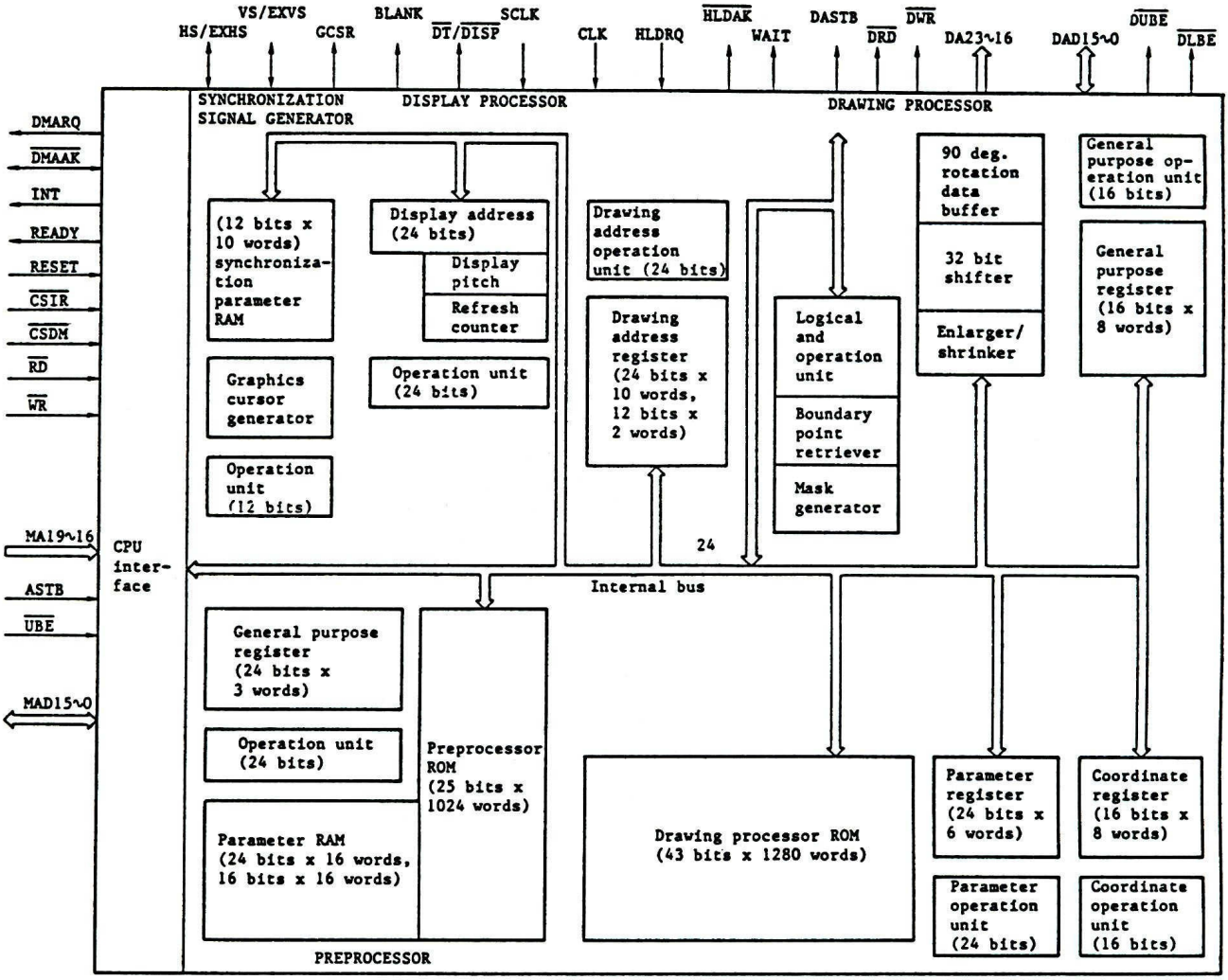


Figure 2 Internal Configuration

The AGDC consists of five functional blocks: synchronization signal generator, display processor, preprocessor, drawing processor and CPU interface. These blocks are connected by a 24-bit internal bus. The feature of this architecture is that the preprocessor and drawing processor perform pipelining, and that the drawing processor is equipped with hardware to perform high-speed painting and enlargement/shrinkage.

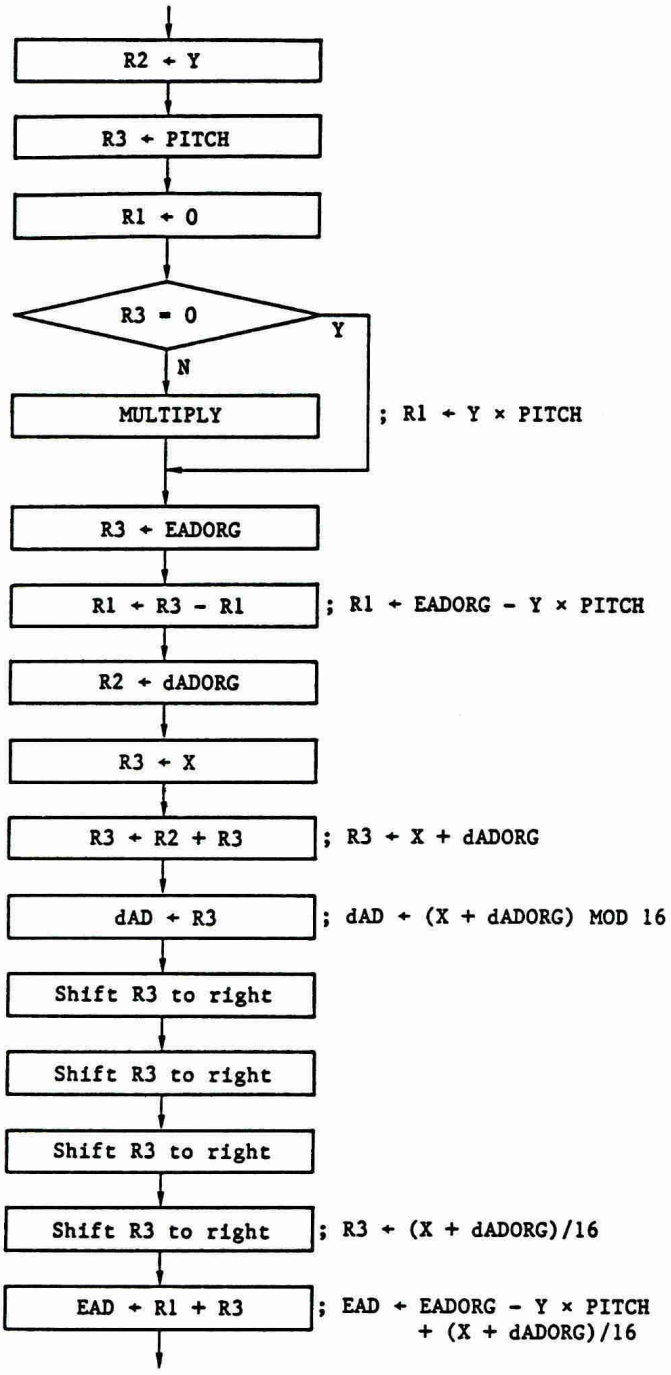


Figure 3 Preprocessor's Processing Procedure to Calculate An Absolute Address From X-Y Coordinates

Each step is executed in one clock cycle (125 ns at 8 MHz). The multiplication (MULTIPLY above) is conducted using the adder-subtractor within the preprocessor. Introduction of the preprocessor has made the pre-drawn processing about 10 times faster than to its previous processing by the CPU.

Preprocessor to Execute Pre-drawn Processing

The GDC had a built-in FIFO to receive commands from the CPU. This was done to adjust the difference between the drawing speed and the pre-processing speed. This FIFO had only a passive role to temporarily store command parameters. With improvements made on LSI integration levels, however, the general tendency now is to let the graphics controller perform pre-processings such as calculation of absolute memory addresses from X-Y coordinates. If this function is added to the drawing processor, it becomes impossible to perform pre-processing and drawing in parallel. So, a decision was made to let the AGDC have a built-in preprocessor to execute parts of the pre-drawn processing at high speeds. In addition, a method to directly write parameters into the register managed by the preprocessor rather than the parameter writing into a passive FIFO was adopted. As soon as a command code was written into the register, the preprocessor starts its operation.

The preprocessor consists of a 24-bit adder/subtractor, a general purpose register, a control ROM and a parameter RAM. (See Figure 2.) The main functions of the preprocessor are as follows:

① Conversion from X-Y coordinates to absolute addresses

② Interpretation of commands

③ Generation of parameters required for drawing graphics

④ Calculation of paint pattern extracted position based on Y coordinate

⑤ Sorting of coordinates used to paint triangles

⑥ Error checking of user-set parameters

⑦ Data exchange with the drawing processor, and start control of the drawing processor

High-Speed Calculation of Physical Address from X-Y Coordinates:

The preprocessor calculates a display memory's absolute address from the X-Y coordinates. Using the formulas shown below, an absolute word address EAD and a dot address dAD are calculated from coordinates (X-Y).

$$EAD = EADORG - Y \times PITCH + \{(X + dADORG)/16\} \quad (1)$$

$$dAD = (X + dADORG) \text{ MOD } 16 \quad (2)$$

EADORG and dADORG show absolute addresses (a word address and a dot address respectively) which correspond to the origin (0, 0) of X-Y coordinates. PITCH shows the number of words of the display memory in the horizontal direction.

Based on the above formulas, the preprocessor executes address calculations with the minimum of hardware at high speeds. For this purpose, we did not adopt an easy method of using a general-purpose processor, because it seemed necessary to design a new optimum processor which is free of unnecessary circuits and can process at high speeds. If the 24-bit adder-subtractor of the preprocessor is used, for example, multiplication between a 16-bit multiplier and a 16-bit multiplicand (whose product is 24 bits long) can be executed with a single clock per bit. To realize this speed, a right shifter is added to a register and a left shifter to another among the three arithmetic registers. And an operation end-detection circuit is added to the adder-subtractor. These devices can also be used to perform integer divisions at high speeds.

Figure 3 shows the flowchart of address calculation. In this figure, MULTIPLY (Y x PITCH) shows the multiplication by the adder-subtractor. When the horizontal resolution of the display screen is 1024 pixels, PITCH showing the number of words is 64 (7 bits actually). Therefore, the multiplication can be executed in 7 clock cycles. And conversion from X-Y coordinates to a physical address is therefore executed in 22 clock cycles which are 7 clock cycles of multiplication plus clock cycles of other steps (in Figure 3).

Hardware Mounted on Drawing Processor for Enlargement/Shrinkage, Boundary Point Retrieval:

The preprocessor is equipped with a single arithmetic unit. And the drawing processor is equipped with more than one arithmetic units, shifters and mask generator. These circuits are controlled to be executed simultaneously within one cycle. A command is 43 bits wide

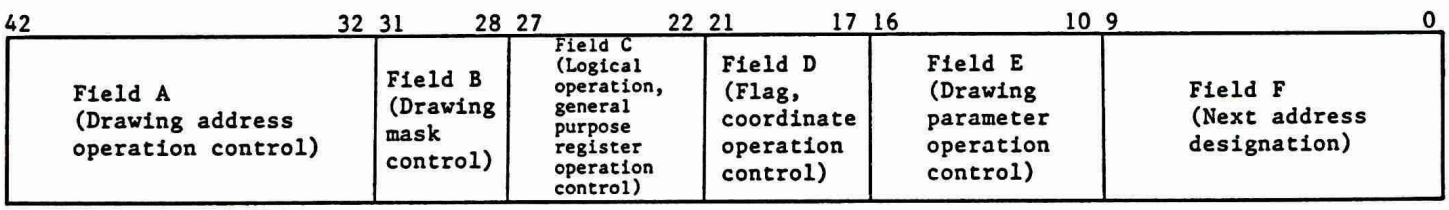


Figure 4 Command Format of Drawing Processor

Consisting of 6 fields (A to F), each command is 43 bits long. Each field can be executed independently of other fields. Because the number of arithmetic units and shifters is larger than the number of fields, a single field can control two or more operations.

consisting of six fields. Each field is assigned a different role of hardware control. (See Figure 4.) A 4-address branch based on judgment results of up to two types can also be executed within a single cycle. Therefore, all address branches and arithmetic commands can be executed in one clock cycle (125 ns). For example, a 24-bit register read, an operation, a rewrite and judgment of the operation all end in one clock cycle.

To achieve the above mentioned processing speed, the control storage is designed so that its asynchronous ROM gives its output as soon as the access address is input. In order to read the command synchronously with the address cycle, a synchronous type temporary storage is added to the command decode output driver. Under this circuit configuration, the control ROM can always be accessed within one cycle.

A tightly connected group of registers is assigned to the 24-bit drawing address operation unit, drawing parameter operation unit, 16-bit coordinate operation unit, and general purpose operation unit of the drawing processor. In addition, the drawing processor is equipped with a display data logic and operation unit, a 90 deg. rotation data buffer, a 32-bit shifter, a data enlarger/shrinker, a drawing mask generator and a boundary point retriever. (See Figure 2.)

To generate coordinates required for a slant copy, a fill-in of a triangle or a trapezoid or a rotational copy of an arbitrary angle, and a fill-in of a circle or an ellipse, a single straight line generator, two straight line generators and a single circle/ellipse generator are used respectively. These coordinate generators are not implemented by fixed hardware wired logic. In fact, only by changing software, it is possible to effectively use the common hardware of the drawing processor in various applications. For example, a straight line generator consists of three parameter registers and one working register.

The AGDC is equipped with functions to draw on the basis of both a pixel or dot and a word. A new drawing mask operator has been developed so that the AGDC can cope with different drawings without any inconvenience. (A clipping generator is also included.) Mask generation and clipping control operate in parallel with other draw processings. So, even when a clipping operation is designated, the drawing speed is not lowered.

Complex Drawings are Partitioned and Pipelined by the Preprocessor and Drawing Processor:

The drawing processor and the preprocessor are each equipped with a coordinate register and a drawing address register. For this reason, when the preprocessor has ended its processing, changing the register contents of the preprocessor does not affect the drawing operation. That is, the preprocessor and the drawing processor can then perform their own processings independently. As a result, while the drawing processor is in operation, the preprocessor can execute the preprocessing of the next drawing.

The preprocessor is equipped with a start designate function as well as a status detect function of the drawing processor. So, a single drawing command can designate an operation of the drawing processor in two or more separate times. For example, drawing a circular sector is actually processed in the following way.

① Calculates drawing parameters of an arc (Preprocessor)

② Draws the arc (Drawing processor)

③ Calculates parameters for drawing a straight line between the end point and the center after a start point and an end point of the drawing have been read from the drawing processor. (Preprocessor)

④ Draws the straight line between the end point and the center. (Drawing processor)
Calculates the parameters for drawing a straight line between the center and the start point. (Preprocessor)

⑤ Draws the straight line between the center and the start point. (Drawing processor)

When drawing an arc or an elliptic arc, the AGDC defines the arc in terms of the coordinates of its start point and end point. In business graphics applications, however, an arc is normally defined in terms of an angle. So, if the trigonometric function for calculating coordinates is simplified the results may designate coordinates which do not exist on the actually drawn arc due to some errors. Even if this kind of erroneous setting is made, the AGDC is ready to compensate the coordinates.

Lets draw an arc in the 1st quadrant as an example. (See Figure 5.) When the slope of the tangent is smaller than 135 degrees, even if the user-set start point does not exist on the actual arc, its Y coordinate at least needs to be equal to that of the actual start point. And when the slope of the tangent is greater than 135 degrees, the X coordinate at least needs to be equal. When the slope is 135 degrees, a small/large comparison using both X and Y coordinates is made (to judge if, in the case of the first quadrant, the drawing position is smaller than the user-set position). As a result, wherever the coordinates are set, the start point or end point of the drawing can be compensated. Therefore, even if the coordinates are designated very roughly, no runaway of the AGDC processing occurs. In the processing flow of the preprocessor and drawing processor described above, the preprocessor reads correct coordinate values in step 3 .

No Hardware Window Keeps the Display Processor Simple

Those parts other than the drawing processor and the preprocessor are explained below. Considering the application records of the GDC, the enlarged display function and screen partition function are removed from the display processor of the AGDC. These functions can partition and scroll the screen at high speeds only by changing the display addresses. But it is also known that these functions have restrained the general use and function of the software in such a way that the scroll range and partition method depend on the size of the display memory mounted. Partially this is why no hardware window has been mounted. As a result, the display processor section has been simplified. The multiwindow display can be realized using the copy function of the drawing processor. And the display processor is equipped with a dynamic RAM refresh function which, during refresh cycles, outputs a 13-bit refresh address as lower bits of the display memory address output.

The synchronization signal generator outputs horizontal/vertical synchronization signals (HS/VS) and a display delete signal (BLANK) according to the user-set parameter values. And it also provides a timing output (GCSR) for displaying the graphics cursor which does not depend on display memory addresses. With the top left corner of the display screen as its home position, the cursor can be displayed at any position. The form of the cursor is either a cross or a block.

The CPU interface controls the timings of the signal exchanges between the system bus and the AGDC. This interface is equipped with an address expansion function which operates when the CPU accesses the display memory, and with an arbitration function of the display bus.

The AGDC has input terminals of two clock systems: CLK and SCLK. The SCLK clock is supplied to the synchronization signal generator and the display processor. The frequency of the SCLK clock is determined depending on the specification of the display unit used and its display resolution. Therefore, it may become necessary to set the clock frequency lower than the maximum operating frequency (fmax) of the controller. If that happens, the drawing speed of the controller cannot be utilized effectively. Therefore, the AGDC is designed so that, different from the SCLK clock, another system of clock CLK can be supplied to the preprocessor and the drawing processor. As a result, the preprocessor and drawing processor can always operate at the maximum operating frequency.

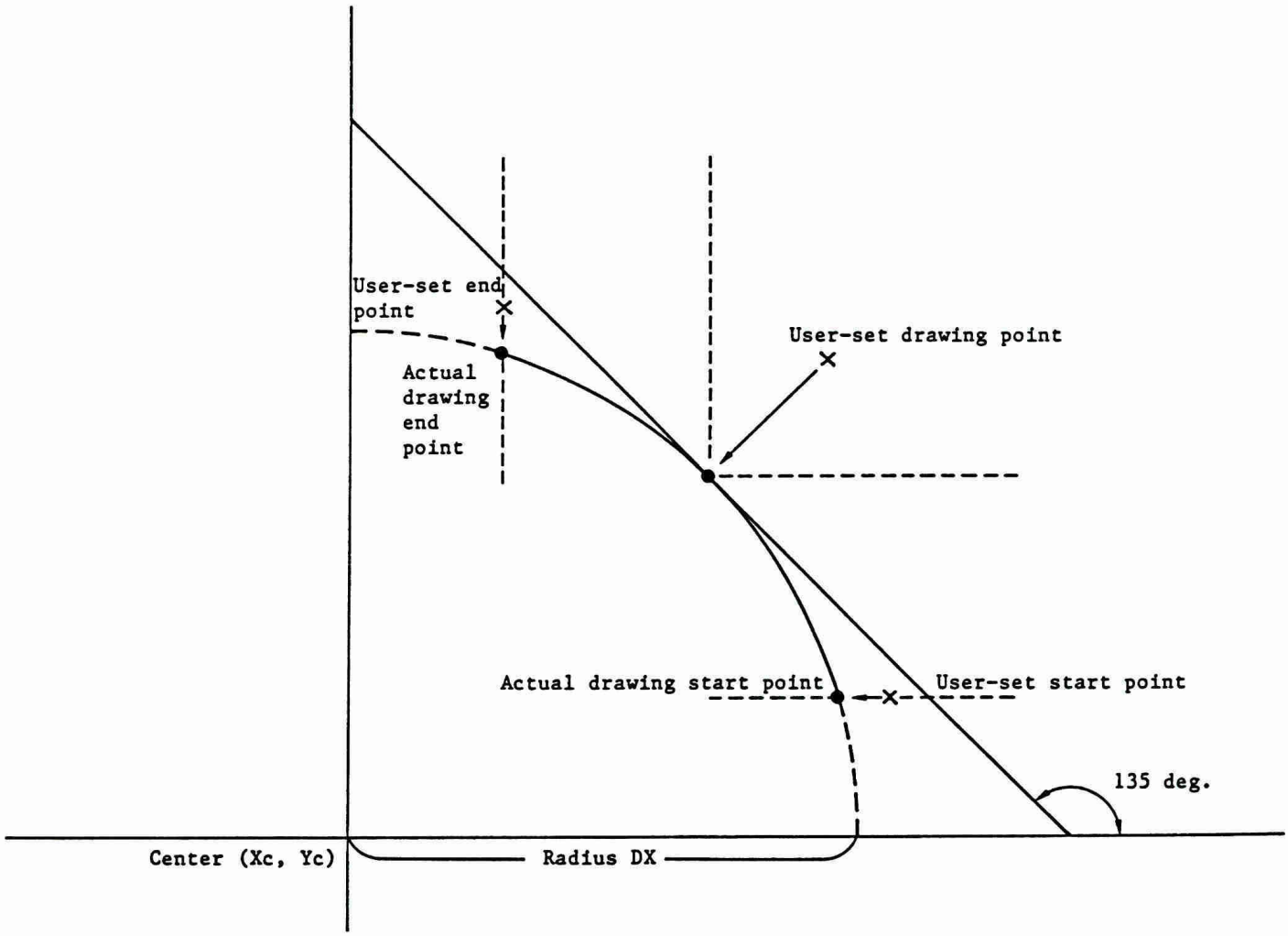


Figure 5 Drawing of An Arc

Given a start point and an end point, the AGDC draws an arc. When the user-designated point does not exist on the actual arc, the AGDC compensates the point.

Up to 32 M Bytes Display Memory Consisting of Dual Port Memory Chips Can Be Controlled

The AGDC directly controls the display memory bus which is separate from the system bus. The AGDC has terminals for outputting a 24-bit word address (DA23-16 and DAD15-0) and inputting/outputting its lower 16 bits (DAD15-0). The capacity of the display memory can be up to 32 M bytes which correspond to 256 memory planes having 1024 x 1024 bits each. It is not necessary to assign all of the memory as the display frame buffer. If the ROM and RAM storing Kanji font patterns and dynamic graphic patterns are mounted as a part of the display memory area, it becomes possible to perform high-speed character font expansion and dynamic graphic control using the copy function.

Kanji font memory (Kanji ROM) can be mounted, as it is, on the display memory bus without attaching anything like an address counter of line direction. Each Kanji JIS code is 14 bits long. And in order to output 32 x 32 dot data onto the 16-bit display memory data bus, it is only necessary to supply 5 row address lines and 1 column address line. That is, if a 20-bit word address (14 bits for Kanji code plus 6 bits for the font) is given, it is possible to mount the first and second level Kanji ROM (of 32 x 32 dot configuration). (The AGDC's word address is 24 bits long.)

In the case of a 16-bit CPU having 20 byte address lines, it is not possible even to directly map the 32 x 32 dot Kanji ROM onto the main storage. If an attempt is made to construct a system such that a character font memory is mounted at the system bus side to transfer font data to the controller, much time will be wasted reading and transferring the font data.

Display Memory is Used to Store Paint Patterns and as a Stack:

With many graphics controllers, each time a drawing is executed, the required paint patterns and character font data are set from the system bus into the built-in register. But because the storage capacity of the built-in register is limited, the data must sometimes be rewritten while drawing or complex paint patterns cannot be selected.

The AGDC can use the display memory as a working area with litle restraint on the storage capacity. Similar to character font and dynamic graphic patterns, paint patterns are stored in the display memory before the painting is executed. And while paint patterns are stored in the display memory, it is not necessary to transfer the patterns. Even when paint patterns are stored in the main storage, they can be transferred using a single block transfer command of the CPU.

The display memory is also used as a stack area for painting data. During the execution of a boundary point retrieval to determine an area to be painted, if a partial closed area has been determined, instead of interrupting the retrieval, the data required to express the closed area are temporarily saved onto the stack. (Details of this operation are described later.) The built-in register may be used for the data stacking but, also in this case, the storage capacity of the register causes an upper limit on the stack level. Although it depends on the algorithm of the boundary point retrieval, as drawing graphics becomes complex, the possibility of the stack level exceeding the upper limit increases.

In order to set a working area in the display memory, it is necessary to give its head absolute address and its size to the AGDC. When a defined stack area overflows, the AGDC stops its operation and informs the CPU of a drawing processor error.

CPU Can Directly Access the Display Memory and the Internal Registers of the AGDC:

For the CPU to be able to access the display memory in the same way as the main storage, it is necessary to map the display memory, as a part of the main storage, on the CPU memory addresses. For this purpose, the AGDC is equipped with address terminals to which 20 byte address lines (MA19-16 and MAD15-0) of the system bus are connected, and with a terminal ($\overline{CSDM}$) which is kept at low level when the CPU accesses the display memory.

The display memory bus has 24-bit word address lines. To extend the byte address of the CPU into a 24-bit word address, an 8-bit bank register is mounted in the AGDC. Figure 6 shows the address extension method. In the extension example, 20H (□) is set in the bank register and a byte address 24DA8H or 24DA9H (□) is given. The display memory lower byte enable signal ($\overline{DLBE}$) (□) is equal to the LSB of the byte address. The data bus width at the system side may be 8 bits. In that case, the upper byte enable signal ($\overline{UBE}$) (□) is always high.

It is possible for the CPU to access the internal registers (i.e. coordinate register and others) of the AGDC. The parameter registers and others of the AGDC can be selected by the 8 lower bits of the CPU's byte address. When the chip select signal ($\overline{CSIR}$) is low, the internal registers can be accessed. Because the display memory and the internal registers under the control of the AGDC can thus be directly accessed by the CPU, it is possible to dump and rewrite their contents using a general-purpose software debugger.

Built-in Bus Arbiter:

In relation to the mapping function, the AGDC has a built-in bus arbiter which gives priority to the use of the display memory bus. It also outputs a ready signal (READY) to request the CPU for a wait operation. Thus, it is not necessary to construct a bus arbiter, as an external circuit, whose timing designs are difficult. Priority sequence for the use of the display memory bus (from high to low) is as follows:

① Generation of display addresses and refresh addresses by the AGDC

② Access of the display memory by other processors with permission of using the display memory bus

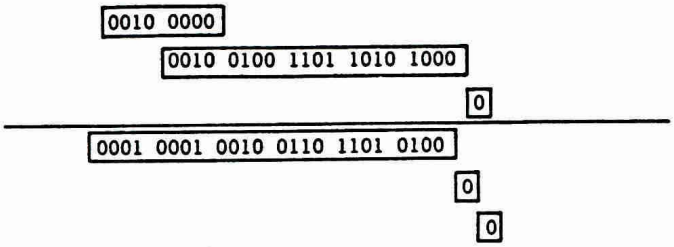
③ Direct access of the display memory by the CPU through the AGDC

④ Generation of drawing addresses by the AGDC

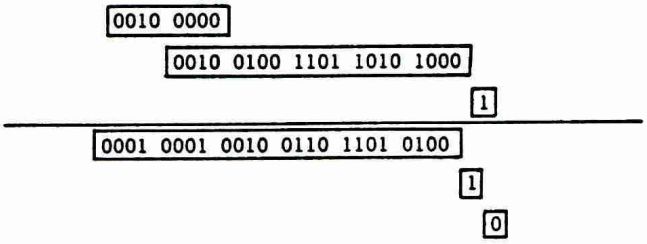
The AGDC generates display addresses with high priority and refresh addresses. For this reason, the AGDC normally operates as the bus master of the display memory bus. Considering the possibility that other external processors will use the display memory bus, the AGDC is equipped with an input terminal ($\overline{HLDRQ}$) for the signal which requests the use of the display memory bus. Accepting this signal, the AGDC puts the address and data output in the display memory bus in floating status and, if some drawing is then in progress, temporarily halts the drawing. And by lowering the acknowledge signal through the output terminal ($\overline{HLDAK}$), the AGDC shows the request source that its request has been accepted. Because the address supply for the display by the AGDC and refreshing are given the

highest priority, the AGDC, by raising the HLDAK signal, reversely requests the right of using the bus to be returned.

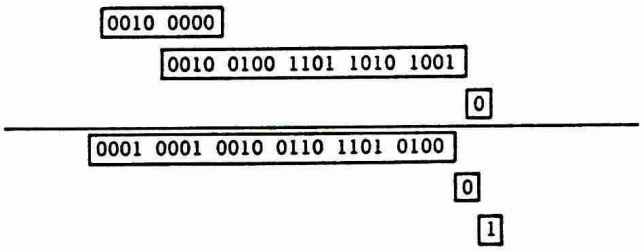
(a) Word access by a 16-bit CPU



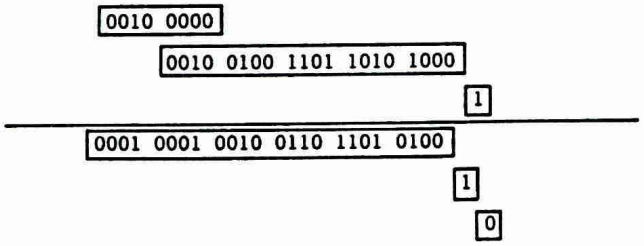
(b) Lower byte access by a 16-bit CPU



(c) Upper byte access by a 16-bit CPU



(d) Access of an even address by an 8-bit CPU



(e) Access of an odd address by an 8-bit CPU

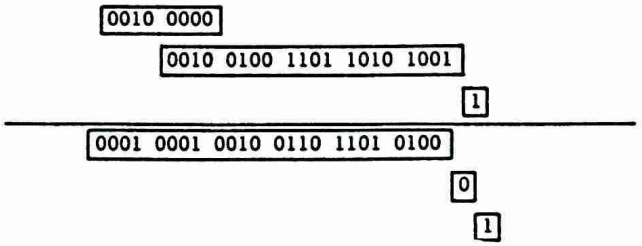


Figure 6 Display Memory Access Method by the CPU

☐ : The content of the built-in bank register

☐ : The byte address input from the system bus

☐ : $\overline{UBE}$ signal which is kept low during an upper byte access through the system bus

☐ : The display memory word address which the AGDC outputs

☐ : $\overline{DUBE}$ signal which is kept low during an upper byte access through the display memory bus

☐ : $\overline{DLBE}$ signal which is kept low during a lower byte access

The address extension examples above show a word access, a lower byte access and an upper byte access by a CPU with a 16-bit data bus width, and a byte access by an 8-bit CPU.

Four Types of Address Cycles are Prepared:

The AGDC offers four display memory bus address cycles as follows (See Figure 7.):

① Drawing address read cycle

② Drawing address write cycle

③ Display address cycle

④ Refresh address cycle

The GDC's drawing address cycle always consists of four clock cycles used for a read-modify-write operation. With the AGDC, one drawing cycle consists of two CLK-synchronized clock cycles, and each display and refresh cycles consists of two SCLK-synchronized clock cycles. If the execution of a refresh operation for dynamic RAM is selected, a refresh address cycle is inserted during the horizontal synchronization period. Two drawing address cycles, read (2 clocks) and write (2 clocks), may be executed continuously or 1 clock cycle of idle time may be inserted between two consecutive drawing cycles.

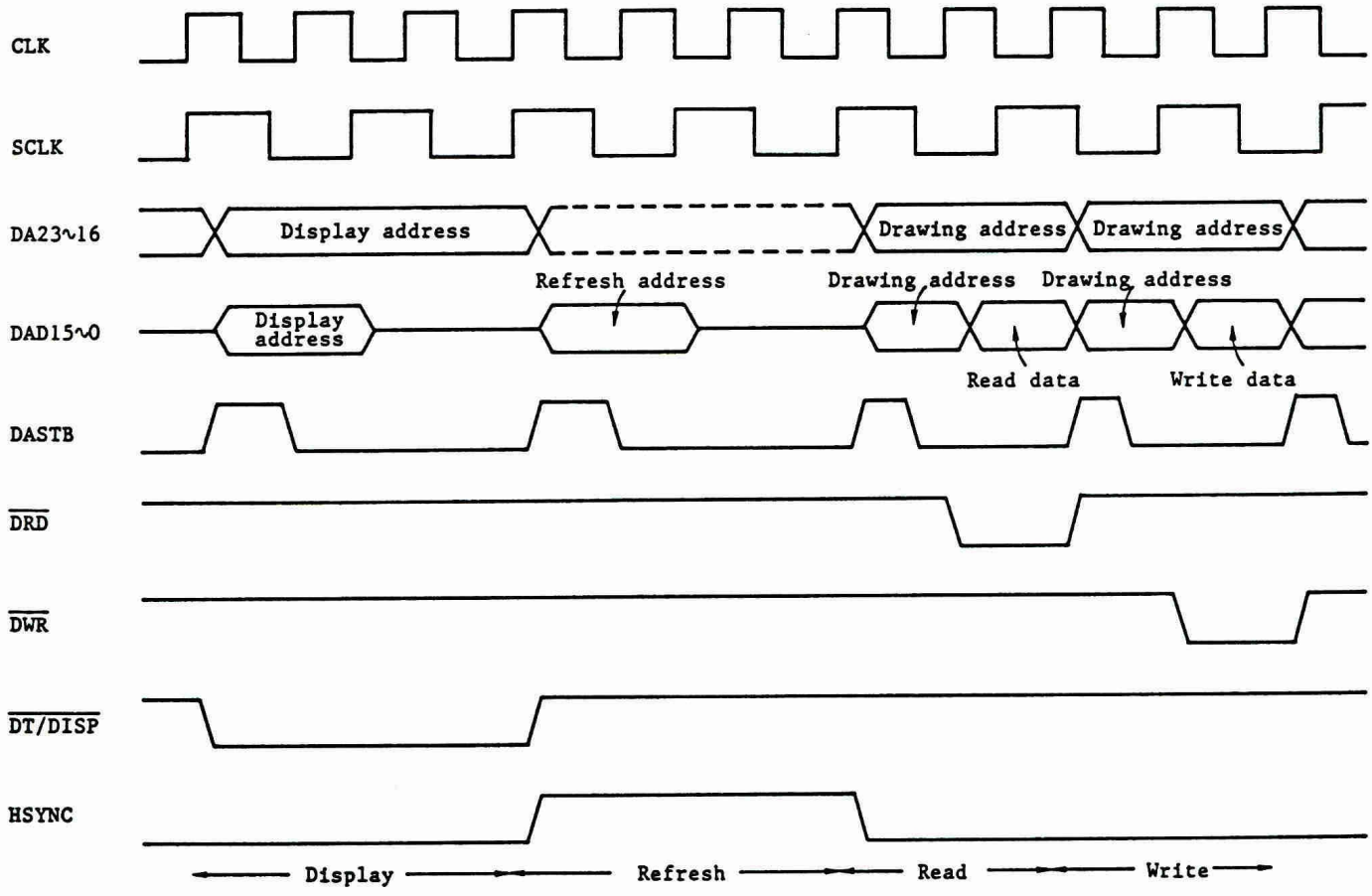


Figure 7 Display Memory Address Cycles

When the display memory is accessed, the address strobe signal (DASTB) is always output. The address strobe signal is synchronized with SCLK during display and refresh cycles, and with CLK during read and write cycles.

Because the address strobe signal (DASTB) is always output at the head of each address cycle, the external circuits can correctly know the address cycle. This DASTB signal is used as the reference timing signal to extract the address alone from the address/data multiplexed terminals or to generate RAS signal and CAS signal supplied to the dynamic RAM.

Because, during the drawing address read cycle, the read signal ($\overline{DRD}$) is output, direction control of the data bus is possible. During the drawing address write cycle, the write signal ($\overline{DWR}$) is output. These signals, after being appropriately delayed by external circuits, are supplied to the display memory. During the display and refresh cycles, only the DASTB signal is output. Whether or not it is a display cycle can be judged by seeing the output signal from the terminal $\overline{DT}/\overline{DISP}$ which is used for both the data transfer timing signal and display cycle timing signal. Whether or not it is a refresh cycle can be judged by the horizontal synchronization signal (HSYNC).

Data Transfer Signal is Output to the Dual Port Memory:

Considering the use of a dual port memory, two drawing timing operation modes are offered as follows:

① Data transfer mode

② Memory cycle steal mode

If a dual port memory is used as the display memory, the data transfer mode is selected. In this mode, clocks of different frequencies may be supplied to the CLK and SCLK terminals. Because of the need by the circuit which synchronizes signals related to the different frequency clocks, the following relationship must be met:

fmax (maximum operating frequency) $\geq$ CLK $\geq$ SCLK

In this status, during refresh cycle and data transfer cycle, no drawing can be performed. With CLK as the unit, during the previous two clock cycles and the following one clock cycle in addition, it may not be possible to perform drawing. (See Figure 8 (a).)

When the same clock signal is supplied to CLK and SCLK, the synchronization circuit can be bypassed by means of a flag setting. In this status, drawing can be performed in all timings except for refresh cycle and data transfer cycle. (See Figure 8 (b).)

If the drawing timing selection is set to data transfer mode, the data transfer timing signal ($\overline{DT}$) is output to the dual port memory. The timing for generating the $\overline{DT}$ signal is selected as follows:

① At the start of displaying each scanning line and when 8 lower bits of the display address become zero.

② At the start of displaying each screen and when 8 lower bits of the display address become zero.

If horizontal direction of the display memory is defined to be larger than the display screen to enable horizontal scrolling, ① above is selected. If the horizontal direction of the display memory is equal to the display screen to perform noninterlace scanning, because continuity of the display address is maintained from the right end of each scanning line to the left end of the next scanning line on the screen, ② above is selected. Because frequency of generating the $\overline{DT}$ signal is lower by ② than by ① , possible timings for drawing increase accordingly. Use of a dual port memory thus enables to cope with a high-resolution display unit whose dot frequency is as high as several hundred megahertz.

As resolution of the display screen becomes high, display data to be read per unit time increases. For this reason, the generation interval of the $\overline{DT}$ signal needs to be shortened. To cope with various display units, from low resolution types to high resolution types, while maintaining high clock frequencies, the AGDC offers eight modes of incrementing the display address, that is, << 1/4, 1/2, 1, 2, 4, 8, 16 and 32 >>. Conceptually, in the case of a 640 x 400 dot noninterlace display, <<1>> can be applied. If the resolution is lower than this, a smaller value is selected. And with higher resolution, a larger value is selected.

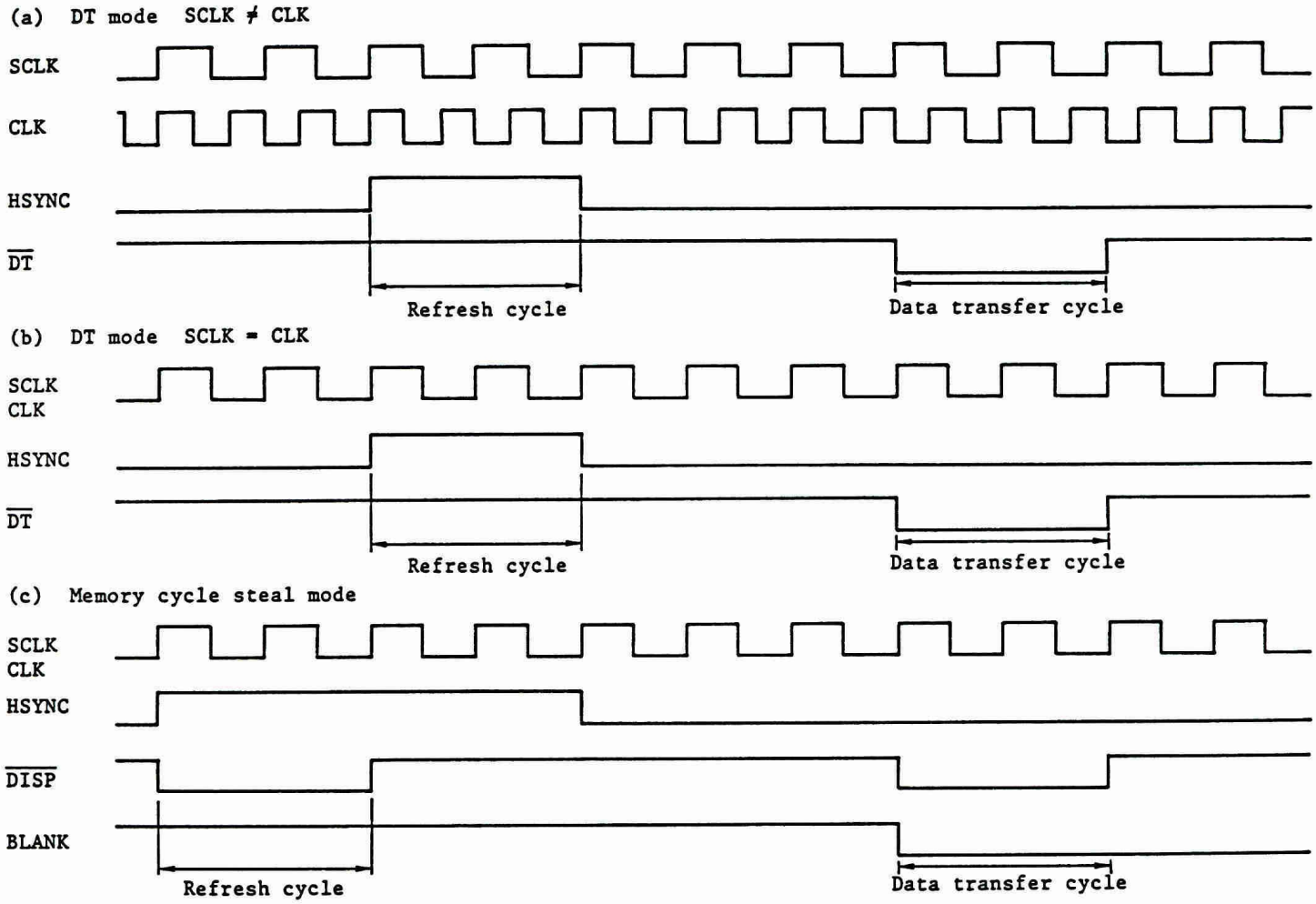


Figure 8 No Drawing - Timings

In those periods indicated by [ ], no drawing can be executed. These timings vary among the DT mode (SCLK ≠ CLK) in which two different clock sources are used, the DT mode (SCLK = CLK) in which the same clock signal is supplied, and the cycle steal mode.

Memory Cycle Steal is Also Possible:

When it is desired to keep the cost as low as possible or when no display is particularly required like a laser printer, normal RAM memory can be used as the display memory instead of dual port memory. For this purpose, memory cycle steal mode is offered. In this mode, the same source for clock signals must be supplied to CLK and SCLK.

During refresh and display periods, a display cycle which includes a refresh address generation and a drawing cycle in which drawing can be performed are generated alternately, 2 clocks by 2 clocks. (See Figure 8 (c).) During these periods, therefore, possible timings for drawing are 50% per unit time. Because, except for the display period, drawing can be executed at all timings excluding refresh cycles, this value increases a little. However, the restraint of memory cycle time determines an upper limit of resolution and, so, this cycle steal mode can only be applied to display units of relatively low resolution. If a standard display unit of 24 k Hz as its horizontal scanning frequency is used, data bus width of the display memory is set to be 16 bits and cycle time of the dynamic RAM used is 300 ns, the upper limit will be on around 640 x 400 dot of interlace display.

Command Interface Uses Parameter Direct Transfer Method

Now that the drawing processor is equipped with high-speed drawing functions, it is important to use them effectively at the system level. That is, the procedure of exchanging commands between the CPU and the graphics controller (i.e. command interface) is seen to determine the actual performance of the system. From the start of development, we fully studied this method.

Generally speaking, command interface methods can be summarized as follows:

① Data transfer method on the system bus

- (a) Parameter direct transfer
- (b) Parameter indirect transfer

② Detection method of the controller status

- (a) Status flag
- (b) Interrupt request
- (c) Wait request
- (d) DMA request
- (e) Hold request

③ Storing method of transfer data

- (a) Indirect address designation
- (b) Direct address designation

④ Storing destination of transfer data

- (a) Register
- (b) FIFO

The AGDC has adopted the combination of ① (a), ② (a), (b) (c), ③ (b) and ④ (a).

Parameter indirect transfer method (① (a)) was not adopted because the data generation time seemed to lengthen the total drawing time (described later). Closely related to this method, DMA request and hold request (② (d), (e)) are also put aside. Regarding the storing method of the transfer data, direct address designation has been adopted. With an indirect address designation method, if adopted, a command and parameters or an address and data must be separated during assembling transfer data. And the number of transferred bytes obviously would increase compared to the direct address designation method by which an address and data can be transferred simultaneously. The storing destination of transfer data has been set to be registers. Even if FIFO method was adopted, it causes an upper limit on the storage capacity because a re-read operation becomes necessary.

The registers in the AGDC are mapped on the CPU's address space. So, for the CPU to give commands to the AGDC, it is enough to execute normal data move commands for the main storage. (See Figure 9.) Those parts indicated by [ ] in Figure 9 correspond to the writing into the parameter RAM in the AGDC. Thirty-two word registers in the parameter RAM are assigned roles respectively corresponding to their addresses. As soon as a command is written, the preprocessor starts its pre-drawn processing.

Three status detection methods are provided (② (a), (b), (c)). These methods may be used being switched as often as required depending on the drawing types. If a system is constructed using a CPU devoted to parameter generation and the AGDC, hardware handshaking using the wait request method (② (c)) is most appropriate. The CPU does not then need to detect the status of the AGDC and can handle accesses to the AGDC in the same way as accesses to the main storage. But when the CPU is in wait status, no interrupt processing can be executed.

If software handshaking by means of status flag method (② (a)) is selected, however, even when the CPU is in a loop being incapable of transferring parameters due to busy status of the controller, interrupt processing can be performed.

Whatever status detection method is adopted, the wait control signal (READY) must be connected to the wait control terminal of the CPU. Occurrence interval of the READY signal differs under various conditions. Take the CPU accessing the display memory as an example. During a read, a wait signal occurs only for five clock periods starting from the fall of the system bus's read signal. And during a write, no wait signal is generated.

```
CHAR:   MOV WORD PTR PITCHS, 0002H; Transfer source address, pitch transfer:  2
        MOV WORD PTR DHH,    0017H; Horizontal direction dot count transfer:  24
        MOV WORD PTR DV,     0017H; Vertical direction dot count transfer:  24
        MOV WORD PTR EAD2H,  0000H; Transfer source upper word absolute address
                                    transfer:  0
        MOV AX, ES : WORD PTR BUF2; Transfer destination Y coordinate
                                    initialization
        MOV BX, ES : WORD PTR BUF3; Transfer destination X coordinate
                                    initialization
        MOV DI, ES : WORD PTR BUF1; Color information initialization
        MOV DX, 0D000H            ; Transfer source lower word absolute address
                                    initialization
        MOV SI, ES : WORD PTR BUF8; Command flag setting
CHAR_2: MOV CX, 041AH             ; Loop count initialization:  26 characters,
                                    4 lines
CHAR_1: MOV PLANES, DI            ; Color information transfer
        MOV EAD2L,  DX            ; Transfer source lower word absolute address
                                    transfer
        MOV X,      BX            ; Transfer destination X coordinate transfer
        MOV Y,      AX            ; Transfer destination Y coordinate transfer
        MOV WORD PTR COM, SI      ; Command, flag transfer
        ADD DX, +30H              ; Character code change
        ADD BX, +18H              ; Transfer destination X coordinate change
        INC DI                    ;
        AND DI, 0007H             ; Color information change
        DEC CL                    ;
        JNZ CHAR_1                ; Is the drawing of 1 line/26 characters ended?
        SUB AX, 0018H             ; Transfer destination Y coordinate change
        MOV BX, 0000H             ; Transfer destination X coordinate change
        MOV CL, 1AH               ; Character loop count initialization
        DEC CH                    ;
        JNZ CHAR_1                ; Is the drawing of 4 character ended?
        DEC WORD PTR BUF7         ; 26 characters/4 lines drawing
        JNZ CHAR_2                ;                                 × 12 times ended?
        ;
```

Figure 9 Part of the Assembler Program Which Expands 1280 Characters (of 24 x 24 Dot Configuration)

The CPU is the 80286. As shown above, command parameters can be easily set only by move commands (indicated by □).

Parameter Indirect Transfer Method Has Problems

Among the parameter indirect transfer methods, there are the receiving command parameters by DMA transfer and (i.e. linked list) the controller itself endeavors to read the command parameter lists which have been generated beforehand on the main storage. By these methods, transfer data strings which include command codes and transfer address information depending on the controller are generated from the original data-like coordinate strings. For this reason, when compared to the parameter direct transfer method which, as soon as coordinate data are generated, transfers them immediately, the data generate processings get in the way. So, operating time of the CPU increases accordingly and software for the parameter transfer becomes complex.

When a series of draw processings is clearly seen through, such as in the case of stroke character drawing which draws characters by means of many straight-line drawings, the indirect transfer method which does not suspend the CPU's pre-drawing depending on the controller's status may be advantageous. But, with DMA transfer method, how to optimize the size of data blocks being transferred becomes a big problem. To fix the size of a transfer block is basically unnatural because no data can be transferred until the fixed amount is ready. And as the transfer block is divided into smaller units, processing time required for the generation of transfer data and operation settings of the DMA controller increases, thus lowering the total drawing speed.

If the controller is of linked list method type to read command lists, this kind of problem does not occur. But the control software will be more complex than that of DMA transfer method because the CPU needs to manage all recognitions of the created command parameter strings and transferred parameter strings, linking to the next parameter strings, and defining the generate positions of the new parameter strings. Setting this information into the controller is also required. And because the controller performs hold control to exchange the fight of using the system bus, this controller may limit CPUs which can be put on the interface.

The Relation Between Parameter Generation Time and Drawing Time Is Important

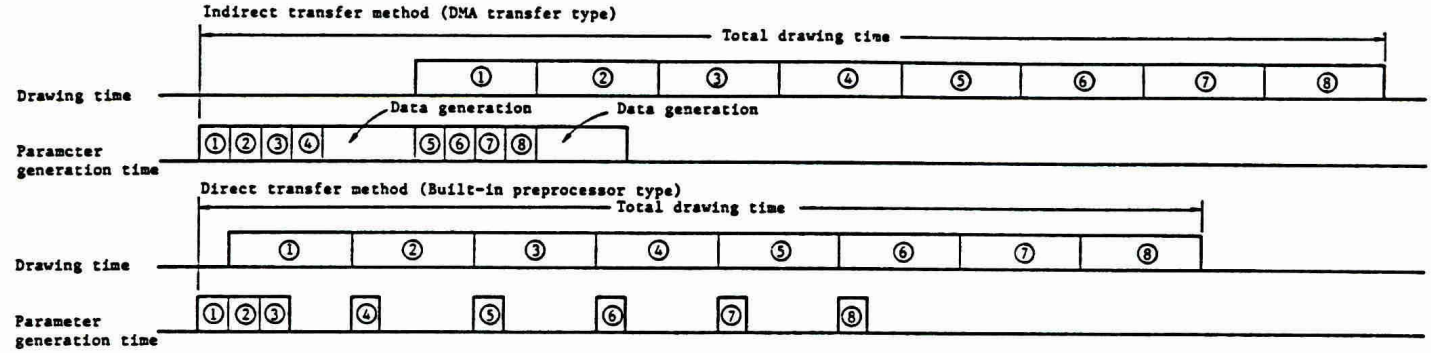
The parameter indirect transfer method aims to execute as far as possible without stopping the CPU processing the parameter generation. In the actual use, however, we judge that it does not have any advantage when compared to the parameter direct transfer method. When comparing these transfer method, the relationship between parameter generation time by the CPU and drawing time by the controller becomes an important factor.

When drawing time is slower than parameter generation time such as in paint drawing, the total drawing time is approximately equal to drawing execution time. But the indirect transfer method is slower than the direct transfer method by as much as the time taken to generate the first data. (See Figure 10 (a).)

If an extreme case of drawing only 1 dot is taken as an example, the total drawing time depends on the parameter generation time. (See Figure 10 (b).) However, as the indirect transfer method generates parameter strings beforehand, the drawing can be executed continuously without depending on the parameter generate processing. This is, of course, limited to the drawing which uses the previously generated data blocks. Although the direct transfer method, as it depends on parameter generation time, cannot continuously draw, the total drawing always ends within a short time.

That is, because the drawing time in a block does not depend on parameter generation time as the direct transfer method does, the indirect transfer method seems faster. In fact, its total drawing time is longer by as much as the time taken to generate data.

(a) Drawing time ≧ Parameter generation time



(b) Drawing time < Parameter generation time

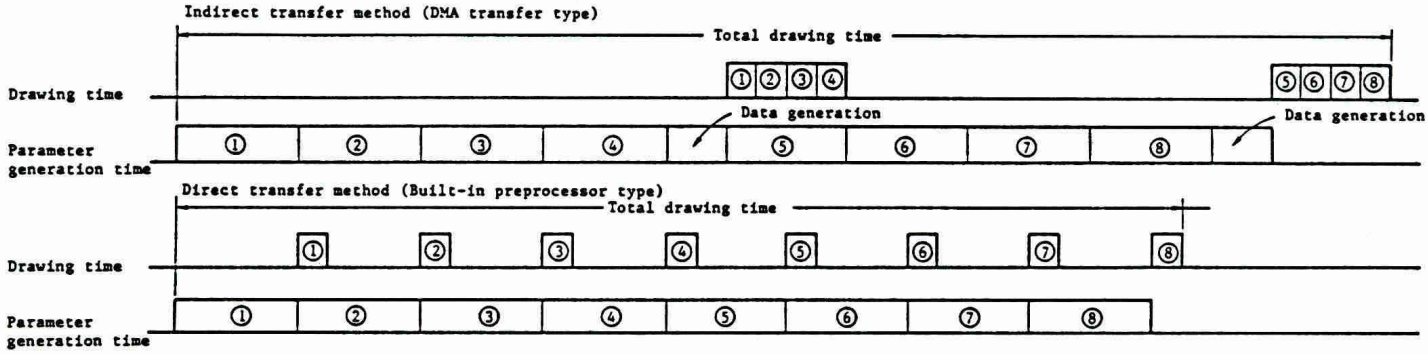


Figure 10 Comparison of Actual Drawing Times

Between parameter direct transfer method and parameter indirect transfer method, the operations of the CPU and the controller are compared. Regarding case (a) in which drawing time is longer than parameter generation time and its opposite case (b), total drawing time is evaluated. Parameter indirect transfer method causes the total drawing time to be longer than direct transfer method by as much as the time taken to generate data.

Drawing Processor Used to Execute High-Speed processings of Enlargement, Shrinkage, Rotation and Painting

Nowadays, we often see display units which offer multiwindow functions in order to display two or more processing results on the same screen and help create new documents while referring to different document contents. Some controllers which enable multiwindow function, display only by changing display addresses without transferring display memory contents. (Ref. 5) This is called hardware window function.

A hardware window operates fast because it does not need to transfer any data at all. To display a window having bit boundaries, however, it is necessary to take display data once into the controller chip. In the case of a display unit with high resolution, a real-time control corresponding to a dot frequency which is as high as several hundred megahertz is required. Because, in general, controllers can only guarantee their operations at frequencies 1 digit lower than dot frequency, this restrains the resolution of display screen. And there is also an upper limit on the number of windows which can be displayed. Although it is desired to increase the number of windows at higher resolution, the number is limited to smaller one on the contrary. This is because all of the change positions of display addresses occurring on each display line and the head display address must be transferred during the horizontal retrace period. Although scrolling in windows then becomes faster, this is also limited by the capacity of the window memory.

Clipping Function is Used for Window Display and Picking

We decided not to mount a hardware window function which is advantageous from the viewpoint of speed but causes considerable restraints during the stage of unit design. The copy function is used instead to transfer display data and form windows. Different control methods are adopted between the pop-up menu whose display contents are fixed and windows. Window control requires a window memory which has the same memory configuration as the frame buffer, but menu display data need not be two dimensional. Only when required, it is enough to expand the original menu data placed on 1-dimensional continuous addresses in a 2-dimensional manner. This fact has enabled to expand 1-dimensional data into 2-dimensional ones or convert 2-dimensional data back to 1-dimensional data.

In order to efficiently form a multi-window, a hardware clipping function is also provided. A rectangle clipping area is defined by designating two diagonal coordinate points. Whether only the inside is drawn, only the outside is drawn, or no clip operation is performed can be set. (See Figure 11.) This function can be used effectively for a window positioned at the front of the display. In the case of painting, definition of this clipping area is used as the outer frame designation of a boundary point retrieve area.

If the clipping function is used, the pick operation specifies and extracts a particular graphic drawn on the screen. If the entire drawing is left to the controller, unless the CPU traces the drawing positions, no pick operation is possible. So, this function is added to the AGDC. Using a mouse or a digitizing tablet, a clipping area is moved to the intended position beforehand and an area as small as 2 x 2 dots is defined. A drawing area is set outside the clipping area, a logic operation mode which does not cause the display results to change is selected, and drawing is executed

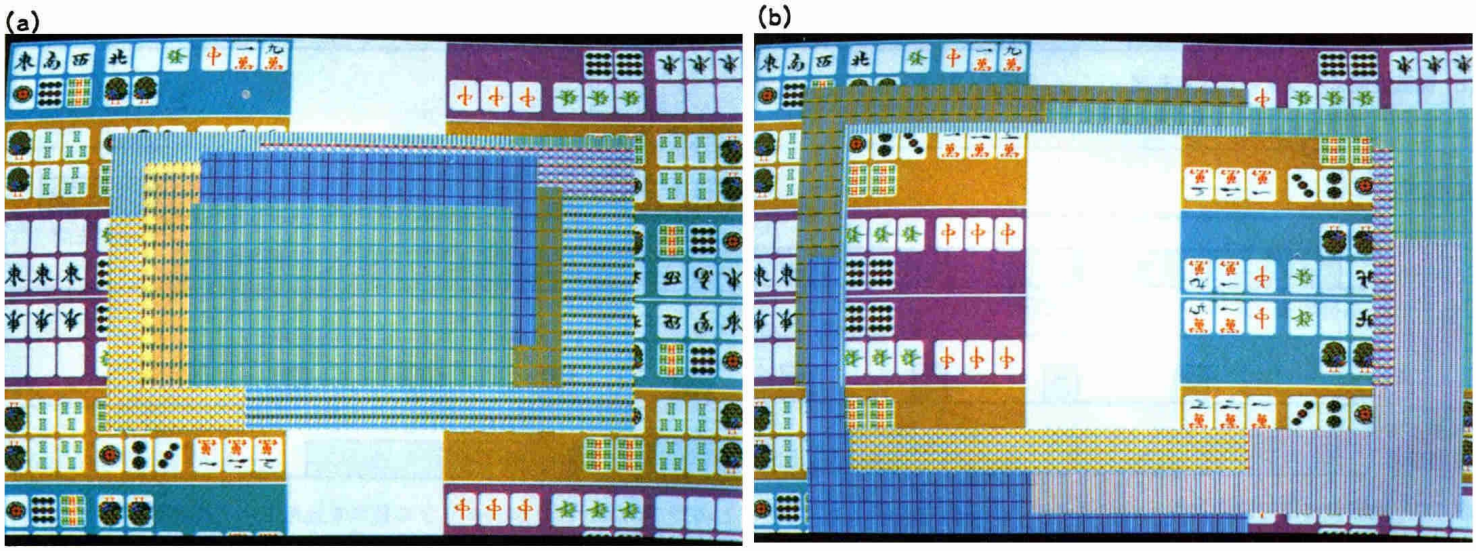


Figure 11 Clipping Examples

Clipping areas are designated, mode of painting inside (a) or outside (b) is selected, and a rectangle fill is executed.

again. By means of an interrupt or a flag, the CPU knows that clipping has occurred and picks the graphic which was then being drawn.

Plane Type and Pixel Type Display Memory Can Be Constructed

Table 1 lists the main drawing functions of the AGDC. But all of their combinations are not necessarily available. (See Table 2.) Functions which are not built in the AGDC, such as drawing of an ellipse whose axes are not parallel to X and Y axes, must be supplemented by the CPU's software. For this reason, if some AGDC-executed functions like clipping are used in other drawings, it is desirable to be able to apply them in the drawing by the CPU. In this case, the CPU can draw the ellipse using the bit mapped method which enables to designate relative positions. In drawing into the frame buffer (in the display memory) which corresponds to the display screen, it is possible to designate positions by means of X-Y coordinates. In data referencing concerning the Kanji ROM and working areas, it is possible to designate positions by means of absolute addresses.

There are two methods to store color information as follows:

① Color plane method

② Packed pixel method

The plane method stores one pixel of color information at addresses a color plane from each other. So, in the case of one-pixel-to-one-pixel drawing like that of a straight line, it is necessary to sequentially execute all planes of drawing bit by bit. But the plane method is suitable for copying, painting, rotating, enlarging and shrinking which would be faster if neighboring bit strings were processed as a whole. Packed pixel method, however, because it packs all pixel information in the smallest access unit (1 word) of the display memory, can access one pixel simultaneously. In graphics drawing which draws one pixel as the unit, the packed pixel method is faster than the plane method. But if the number of dots constituting one

pixel is changed, it will cause such hardware as drawing flow and drawing mask to change, thus being less flexible.

The AGDC can apply all of its drawing functions to the plane method of display memory. In the case of bit mapped drawing, the AGDC can cope well with the pixel method of display memory. Because this mode can be dynamically selected between plane method and pixel method during drawings, it is possible to mount a display memory with two configurations (plane method and pixel method) to make full use of their respective features.

Unlike the GDC, the AGDC executes color drawing by issuing a single drawing command even for plane method of display memory. In graphics drawing or painting, up to two logical operations can be designated for two or more planes at the drawing destination. With four memory planes, for example, a different logical operation can be executed for every two planes. In copy and put/get operations, similar processings can be executed for transfer source and transfer destination.

It is easy to output a hard copy of the color screen to a dot-matrix color printer. This is done by executing logical operations appropriate for each ink ribbon among the transfer source data which are stored in two or more planes and issuing once a 90 degree rotation get command which reads 90-degree rotated data into the system bus. This operation takes out the dot strings (from the AGDC) which are used to directly drive the print head pin for 16 dots vertically and 1 line horizontally.

Table 2 Possible Combinations of Drawing Functions

	Pixel configuration	Hardware clipping	Selection of enlargement, shrinkage and line type	Selection of line type	Selection of paint pattern	Transfer destination (drawing destination) position designation	Transfer source (line type, paint pattern) position designation
Dot	O	O	x	O	-	Coordinate	Built-in register
Straight line	O	O	Enlargement only	O	-	Coordinate	Built-in register
Rectangle	O	O	Enlargement only	O	-	Coordinate	Built-in register
Circle, arc	O	O	x	O	-	Coordinate	Built-in register
Ellipse, elliptic arc	O	O	x	O	-	Coordinate	Built-in register
Paint	x	O	-	-	O	Coordinate	Built-in register, head address
Rectangle fill	O	O	-	-	O	Coordinate, absolute address	Built-in register, head address
Circle fill	x	O	-	-	O	Coordinate	Built-in register, head address
Trapezoid fill	x	O	-	-	O	Coordinate	Built-in register, head address
Triangle fill	x	O	-	-	O	Coordinate	Built-in register, head address
Normal copy	O	O	O	-	-	Coordinate, absolute address	Coordinate, absolute address
90 deg. rotation copy	x	O	x	-	-	Coordinate, absolute address	Coordinate, absolute address
Slant copy	x	O	x	-	-	Coordinate, absolute address	Coordinate, absolute address
Arbitrary angle rotation copy	x	O	O	-	-	Coordinate, absolute address	Coordinate, absolute address
Normal put/get	O	O	x	-	-	Coordinate, absolute address	-
90 deg. rotation get	x	O	x	-	-	-	Coordinate, absolute address
CPU direct drawing	O	x	x	-	-	Coordinate	Arbitrary

Enlargement Factor of 16/N, Shrinkage Factor of N/16:

Enlarged and shrinked copy operations are set to 16/N for enlargement and N/16 for shrinkage (N is an integer from 1 to 16). The fact that many factors can be selected near the original size (i.e. factor of 1) is practical. The restraints on enlargement and shrinkage factors were required to enable simple hardware to achieve high-speed enlarging and shrinking copy operations. Let's take 15/16 reduction as an example. This is just done by extracting one predetermined bit from among the 16 bits read from the display memory. In cases of shrinkage the extracted bits are predetermined depending on the shrinkage factors.

Shrinked data are written again into the display memory after being processed through shift, mask and other operations. In the case of a last word, the number of bits whose enlargement or shrinkage has not been completed may not be 16 bits. In this kind of processing, drawing mask operation is difficult. The AGDC, based on the factor and the number of bits whose drawing has not ended, generates masks by referring to the table in which mask formats have been entered beforehand.

Because vertical enlargement and reduction are not sensitive to processing speeds, they are handled on a count basis by software. For doubling, the same line is simply drawn twice and, for shrinkage, lines are just thinned out. In graphics dealing with characters and figures, thinning a single line and leaving no gaps may cause visual information to drastically decrease. It now seems necessary to consider some sort of supplementary processings for shrinked data depending on their applications, such as a straight line drawn not to delete a ruled line.

Three reference methods for paint patterns are shown below.

① Built-in register reference

② Display memory reference

(a) Patterns common to all planes are referenced.

(b) Independent patterns of each plane are referenced.

① is the so-called "solid patterns" which are repeated in vertical direction and in plane's direction. This will be used in clear operation. If reference method ② (b) is adopted, as different patterns can be referenced in each of the horizontal, vertical and plane directions, painting a different color at each dot (i.e. tiling) becomes possible. The AGDC is equipped with functions used to calculate initial values of and change the reference positions of paint patterns, depending on the Y coordinates of the line being painted.

High-Speed Boundary Point Retrieval Leads to Fast Painting:

As far as painting of an area with the same size is concerned, filling is faster than painting. Coordinates used to execute a fill-in can be generated at high speed. And in painting, to specify the area to be painted, read contents of the display memory and retrieve boundary points are necessary. For speeding up painting, to end this boundary point retrieval in a short time is a task of the highest priority. Immediately after 16-bit retrieve data are given, our newly developed boundary point retriever can return the following information to the drawing processor.

① Does a boundary point exist?

② If a boundary point exists, where is its dot position?

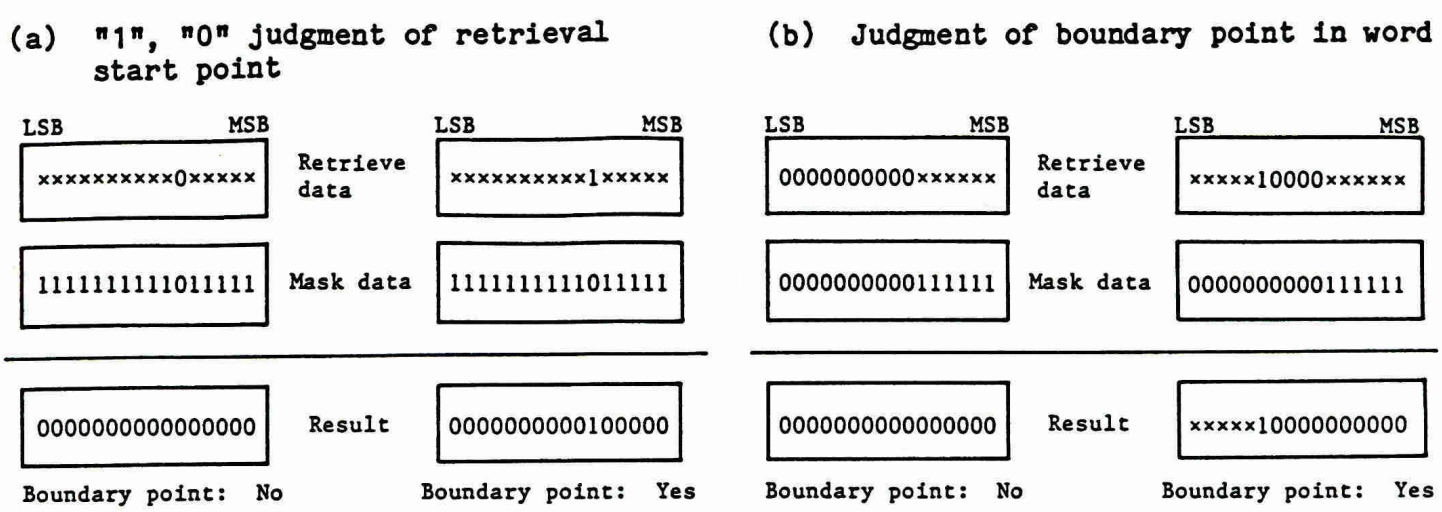


Figure 12 Judgment of Boundary Point

From the result obtained by masking retrieve data, it is known whether a boundary point exists in the word or not. If it exists, its dot position is known. In the above figure, "X" shows an arbitrary value ("0" or "1").

If no boundary point exists within the retrieve data, the content of the next neighboring word is read and, from there, retrieve data are created. In the case of a color display, the boundary point is given as a boundary color. If there are three planes, data of the three planes are continuously read, two clock cycles per plane, and logical operations are executed between the plane data according to the boundary color designation, thus generating retrieve data. When reading the data of all the planes has ended, generation of 16-pixel (word) retrieve data ends.

To specify an area as a retrieving object, it is possible to mask the retrieve data. (See Figure 12.) In Figure 12, only those bits for which mask data are "0" are retrieving objects. To judge whether or not the retrieval start point itself is included within the boundary, mask data are given so that only the retrieval start point's data are taken out ((a)). If the result is "0", it shows that no boundary point exists in the word. When continuing to retrieve to the left or to the right from a point within the word, those retrieve data from the point to the word boundary are masked so that the once-retrieved position will not be a retrieving object again ((b)). When there are many boundary points in the word, it is just enough to change only the mask for the same retrieve data. Position data output from the boundary point retriever is used not only in calculating the size of a painting area but in generating retrieve mask data within the word.

Upper and Lower Lines are Read, and Sneak Points of Area are Extracted:

Figure 13 (a) shows the sequence of painting a closed area and how its boundary area data are pushed and popped. The retrieval starts from a position marked X and the area is painted in the sequence from ① to ⑭.

By the sneak points, the closed area is divided into ⑭ smaller areas. The arrow by the side of each number shows the direction of painting to be executed.

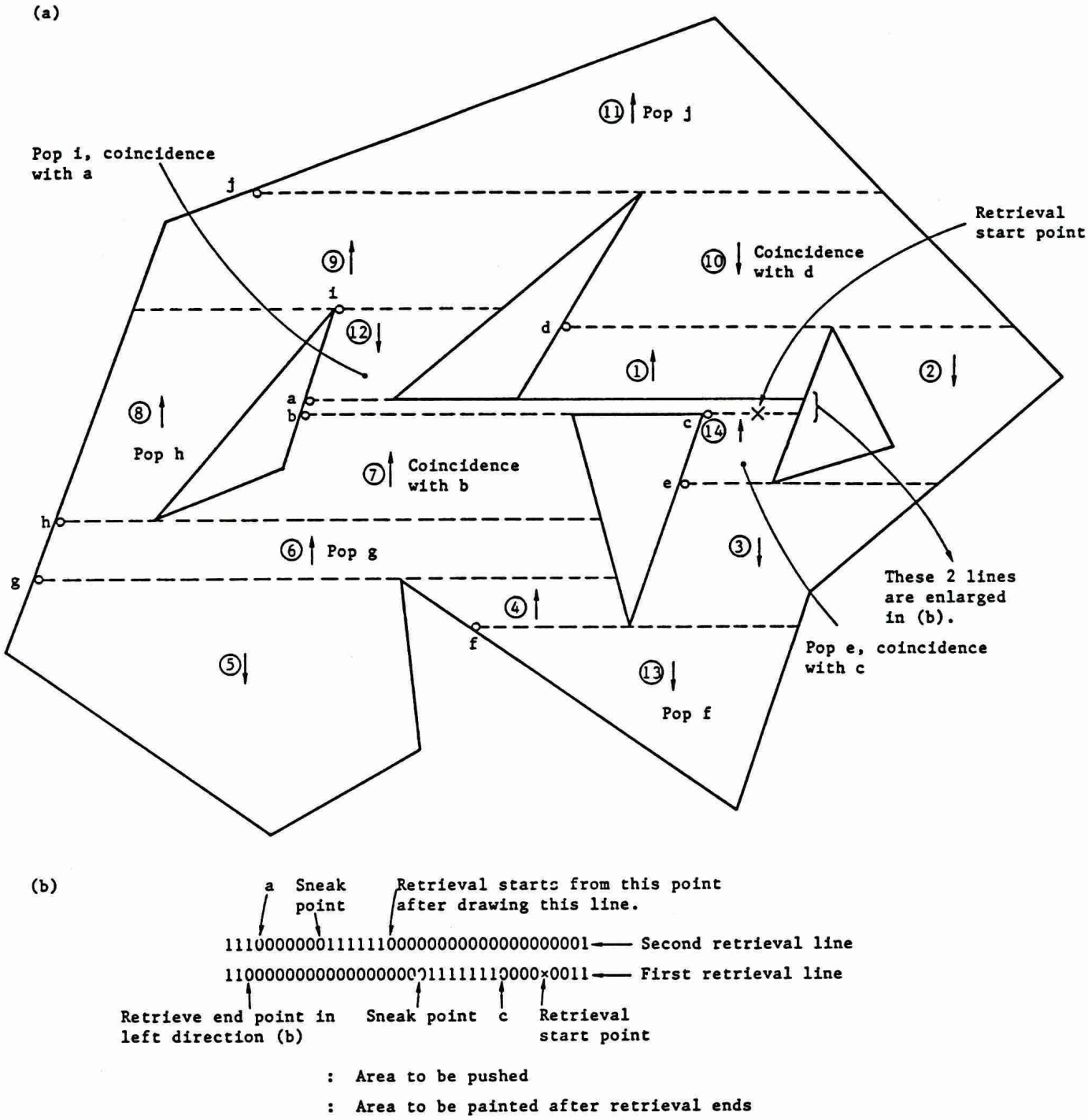


Figure 13 Paint Operation Procedure

An arbitrary closed area is divided into two or more smaller area and painted ((a)), from ① to ⑭, in the direction shown by each arrow. A part of the closed area in (a) is enlarged in (b) to show the procedure of retrieving boundary points. The boundary point retrieve processing includes specifying a closed area based on retrieved change points from "0" to "1" or from "1" to "0", extraction of sneak points, retrieval-end judgment, push and pop of the closed area information and actual painting inside the closed area.

Painting which includes boundary point retrieval proceeds in the following way. Retrieval starts from the retrieval start point shown in the figure. The line where the retrieval start point lies and its upper line are retrieved at the same time. When, during the retrieval of one line, boundary points to the left and right are found, those parts' information (i.e. closed area information) are pushed onto the stack. In this example, a, b and c are pushed. In area ①, painting is executed from ___ and operation proceeds to retrieval of its upper line. When, at the end of the retrieval in ①, a sneak point into ② occurs, d is pushed and painting of ② starts. While the execution of painting thus continues, closed area information is piled on the stack in the display memory.

When painting of ⑤ has ended, a, b, c, ..., g have been pushed onto the stack. In area ⑥, the stack pops and retrieval and painting start from g which was pushed at the end of retrieval in ④. And in ⑥, h is pushed later.

When post-retrieval painting is executed, it is always checked to see if the closed area information specified by retrieval coincides with the stack contents. In painting of ⑦, as the coincidence with b is obtained, flag is raised at b's closed area information so that no retrieval will be executed again after popping.

In the boundary point retrieval described above, the display memory contents of two (upper and lower) lines are read alternately. This method enables to make clear the flow of boundary point retrieve processing (Ref. 7), that is, extraction of sneak points around the area to be painted, stack operation when the area is determined during the retrieval, and retrieval-end judgment. Two lines of data are read and the results of respective boundary point retrievals are compared to retrieve boundary points. When an area to be painted is determined during retrieval, as described before, six words of information including position and size of the area is pushed onto the predetermined stack area on the display memory.

A part of the figure (a) is enlarged in part (b) of Figure 13. First, that the retrieval start point itself is not included within the boundary is confirmed and, then, the change point from "0" to "1" is retrieved. As the result, point c is obtained. Next, boundary point judgment is made on the point one line higher than c. If the point is "1", as it means that the boundary in the left direction is found, retrieval proceeds to right. In this example, "0" is first obtained, so retrieval proceeds to left. When leftward retrieval starts, change point from "1" to "0" on the first retrieval line and from "0" to "1" on the second retrieval line is retrieved.

Generally speaking, a point changing from "0" to "1" becomes a boundary point and, if its upper or lower point is "1", the point is the retrieval end point. With "0", it may be a sneak point. Therefore, a point of change from "1" to "0" in further is retrieved. The processing continues until the retrieval end point is found.

After the retrieval end point b in left direction is found, rightward retrieval starts. Then, during retrieval, the closed area information of a and b have already been pushed. During the rightward retrieval, the closed area of c is first determined and its information is pushed. When the rightward retrieval has ended, the line from which painting is executed is determined. With the left end point of this line as the retrieval start point, retrieval for the rest of area ① begins.

90 Degree Rotation Is Executed Using the Buffer Register:

The built-in 16-word buffer register is designed to speed up the 90 degree rotation operation. This buffer is structured to read rows written data is columns, by the address bus and data bus connection. In addition, a flow control is adopted so that, by judging the mutual relationship between the bit positions of transfer source areas and those of transfer destination areas, no garbage occurs during the data transfer to the buffer. This buffer also functions as an FIFO to temporarily store the transfer data in get/put operations.

Arbitrary angle rotation copy which is accompanied by enlargement/reduction is executed by means of straight line drawing. Generally, when a figure is rotated, some points of no drawing appear in the figure drawing area. If the rotation angle with relation to an axis reaches 45 degrees, this phenomenon is usually noted. Judging this kind of singular points, it is possible to select whether or not drawing of the neighboring points are also executed similarly for these singular points.

Evaluation of Execution Speed by Various Benchmark Tests

The evaluation method of drawing speeds of graphic controllers differs very much among different manufacturers. The environmental conditions for evaluation are often not very clear. Therefore, it is difficult to compare the drawing speeds of different controllers simply based on their specifications. The performance of some controllers is expressed in such a way as "some nanoseconds per dot" or "some vectors per second". But these values are something like the maximum instantaneous wind speed, which does not consider such system factors as time ratio showing how long the display memory can be occupied for drawing or command interface. Therefore, the actual drawing speed often comes to differ considerably.

Values useful for unit designers who judge if particular controllers can be suitably adopted, are the drawing speeds available when the intended systems are established. For that purpose, the method of calculating the number of straight lines or the number of characters which can be drawn in one second based on the time taken to draw a single straight line or a single 24 x 24 dot character is not good either. Because, when drawing operations are continuously executed, how far the respective drawing operations are from each other is not included in the evaluation method.

Table 3 Number of Drawing Cycles

Drawing contents	No. of drawing cycles
Straight line	4 clocks/dot (pixel)
Horizontal straight line of rectangle (of plane configuration)	4 clocks/word
Circle, ellipse	6 clocks/dot (pixel)
Copy	
Data replacement alone	4 clocks/word
Logical operation execution	6 clocks/word
Fill	
Built-in register reference, data replacement alone	2 clocks/word
Built-in register reference, logical operation execution	4 clocks/word
Display memory reference, logical operation execution	6 clocks/word

The total drawing speed should not be based on these values alone. For painting, the number of cycles varies depending on the reference patterns. However, it will affect little the total paint drawing speed.

We decided, in cases of drawing graphics and graphic characters, to evaluate an overall drawing speed available when the same type of drawing is continuously executed. With coordinate values given first, we actually measured, using a logic analyzer, the time taken by the CPU from its starting of pre-drawn processing to its ending of the last drawing. In the case of controllers using the parameter indirect transfer method, the expressed time includes the creation and processing of DMA transfer data and command lists. Get and put which depend on the CPU's data transfer speed are excluded. Dual port memory chips are used as the display memory and, as its refreshing consumes 6%, 93.7% of possible drawing timings are secured.

One Word Processing in 2 to 6 Clock Cycles

Before showing the measured values, Table 3 lists the number of clock cycles (i.e. theoretical value) required for the execution of a drawing. A straight line is drawn in the same number of clock cycles as with the GDC, but circle and ellipse are rather slow. This is because the drawing algorithm of an ellipse is based on circle, and that an entire circle can be drawn by a single command. When drawing a rectangle for the plane configured display memory, 16 dots can be drawn in one drawing cycle on the horizontal line. A copy which can designate both a transfer source and a transfer destination on bit boundary is executed at the speed of 6 clock cycles per word while executing logical operations for the transfer source or transfer destination. If the existing data at the transfer destination are ignored and replaced by the transfer source data, 1 word of copy ends in 4 clock cycles.

As for painting, execution cycles differ between the case of referencing paint patterns stored in the built-in register and the case of referencing patterns on the display memory. Compared to the built-in register reference, referencing patterns on the display memory needs more time (as much as 2 read clock cycles). As for rectangle painting, if the setting of replacement alone is made similarly to the case of copy, 16 bits can be painted in 2 clock cycles.

Fast Paining of Arbitrary Closed Area

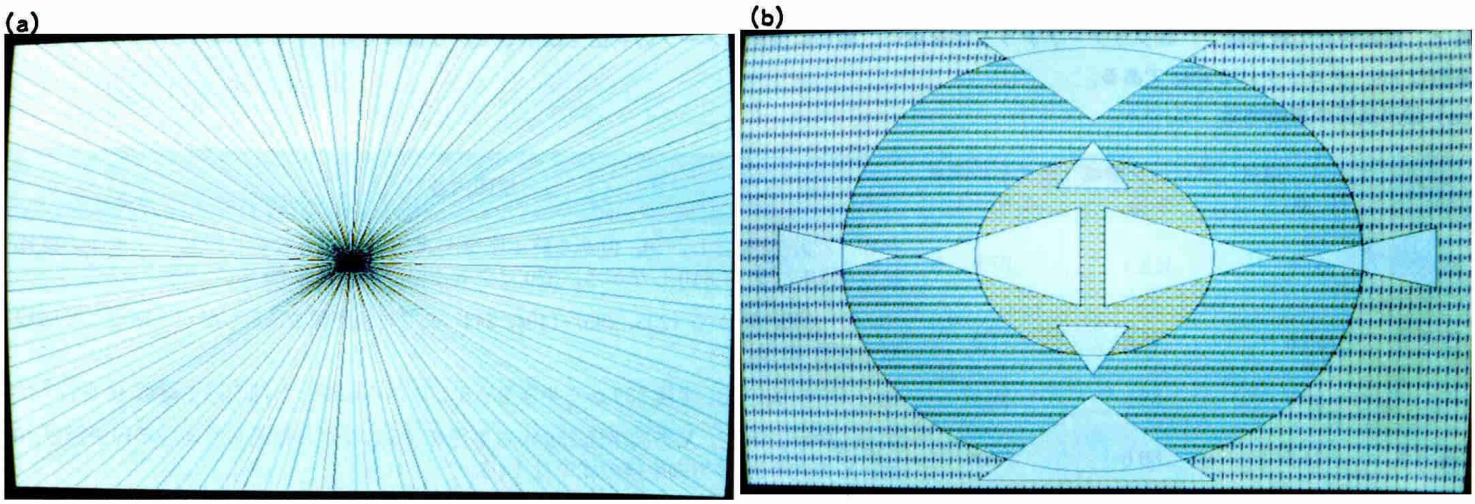
Execution times of graphic drawing, painting and copy are shown in Table 4 (a), (b) and (c), respectively.

If the drawing time of one display memory plane alone is compared with the drawing time of 3 planes (8 colors), in the case of straight line (Figure 14 (a)), it increases three times. In the case of circle and ellipse, however, it increases to a little over two times (Table 4 (a)). This is because, among 6 clock cycles required for drawing a circle or an ellipse, only the execution time of 4 clock cycles needed to draw 1 plane increases three times to 12 clock cycles.

Screen clear shown in Table 4 (b) is an example of painting. By referencing the paint patterns stored in the built-in register, the rectangle is painted. If the left side and right side of a rectangle are bit boundaries, painting of the word requires mask processing and a read-modify-write operation is executed. But, this screen clear has no bit boundary and 115,200 words are executed within 28.8 ms. This fact shows that 1 word is cleared in 2 clock cycles. Painting speed, except for that

Figure 14 Benchmark Processings Used To Measure Drawing Times

(a) Straight lines (in Table 4 (a))
(b) Paint (in Table 4 (b))



of rectangle depends little on the reference method of paint patterns. Although read time differs depending on the reference method of paint patterns, this is because it takes long to calculate coordinates for securing paint areas and retrieve boundary points.

Arbitrary closed area painting is very fast (Table 4 (b)). As shown in Figure 14 (b), about 80% of the screen is painted. However, it is a little less than two times compared to the time taken to copy the whole screen consisting of three planes to another three planes. Copy just does the calculation of simple drawing addresses, but paint includes such complex processing as boundary point retrieval. This speed is the result of the previously described hardware to execute boundary point retrieval at high speeds and the retrieval algorithm. And there is not much difference in the execution time of drawing between 90 deg. rotation copy and normal copy. It can be assumed that enlargement and shrinkage take about two and three times as much time as normal copy, respectively.

Drawing time of expanding the 24 x 24 dot character font has been evaluated separately from the copy of a large area. The font is drawn not on word boundary but on bit boundary. There is little difference of execution time observed between slant expansion which calculates drawing start coordinates using a straight line generator and normal expansion. This proves that parallel processing and pipeline processing are conducted efficiently. Calculating from the evaluation results of this continuous graphic character drawing, it is understood that 13,640 characters in black and white or 8,040 24 x 24 dot characters in 8 colors can be drawn in one second. This value seems to be fairly low compared to the one which is calculated based on the number of drawing clock cycles, but is a practical figure.

Table 4 Measurement Values of Drawing Time

Using a logic analyzer, the time taken from the first generation of drawing parameters to the ending of drawing is measured (unit : ms, clock frequency : 8 MHz).

(a) Graphic drawing time

	Plane configuration : 1 plane	Plane configuration : 3 planes	Pixel configuration : 4 bits	Drawing contents
Straight line	37.8	113.4	37.8	With (319, 239) given as the start point, 128 240-pixel straight lines are drawn with their end coordinates starting from (0, 0), which is followed by ± 10 in X direction. And 96 320-pixel straight lines are drawn with ± 10 in their end coordinates in Y direction.
Rectangle	15.6	46.8	37.8	From (0, 0) - (639, 479) to (235, 235) - (404, 244), 48 rectangles are drawn with their diagonal coordinates changing at the rate of ± 5.
Circle	30.0	69.0	30.0	With (319, 239) as their fixed center, 46 concentric circles are drawn with their radii changing from 239 at the rate of -5.
Ellipse 1	34.8	80.6	34.8	With (319, 239) as their fixed center, 46 concentric ellipses are drawn with their X-direction radii changing from 239 at the rate of -5. Ratio of their X-direction radii to Y-direction radii is 4 : 3.
Ellipse 2	45.0	104.2	49.8	The drawing conditions are the same as above (ellipse 1) except that the ratio of their X-direction radii to Y-direction radii is 3 : 4.

(b) Paint drawing time

	Solid pattern	Tiling 1	Tiling 2	Drawing contents
Screen clear	28.8	-	-	Clearing of 640 x 480 dots x 3 planes x 2 sets (115,200 words).
Rectangle fill	7.1	13.5	13.8	Logical operation-added painting of a 400 x 300 rectangular area.
Circle fill	9.3	9.3	9.3	Logical operation-added painting of a circular area whose radius is 150 dots.
Ellipse fill 1	6.7	6.7	6.7	Logical operation-added painting of an elliptic area whose X-direction radius is 150 dots and ratio of X-direction radius to Y-direction radius is 4 : 3.
Ellipse fill 2	12.3	12.3	12.3	Logical operation-added painting of an elliptic area whose X-direction radius is 150 dots and ratio of X-direction radius to Y-direction radius is 3 : 4.
Triangle fill	3.0	3.0	3.0	Logical operation-added painting of a triangular area which is enclosed by 3 points of (152, 419), (320, 240) and (459, 320).
Trapezoid fill	5.2	5.2	5.2	Logical operation-added painting of a trapezoid area which is enclosed by 4 points of (0, 150), (300, 150), (30, 0) and (270, 0).
Paint	94.5	96.7	97.8	Painting of any of the 3 closed areas formed by 2 circles whose center is (320, 240) and radii are 100 and 220, a rectangle of (0, 0) - (639, 479) and 8 triangles.

(c) Copy drawing time

	1 plane → 1 plane	3 planes → 1 plane	1 plane → 3 planes	3 planes → 3 planes	Drawing contents
Normal copy 1	12.0	29.2	24.3	36.3	No logical operation-added transfer of a 640 x 480 dot rectangular area.
Normal copy 2	18.0	29.2	40.5	52.1	Logical operation-added transfer of a 640 x 480 dot rectangular area.
90 deg. rotation copy	20.2	29.2	43.1	61.1	Logical operation-added, 90 deg. rotation-accompanied transfer of a 480 x 640 dot rectangular area.
Enlarged copy	34.1	-	56.6	102.3	Logical operation added transfer of data enlarged by a factor 16/13 to a 640 x 480 dot rectangular area.
Shrinked copy	63.3	-	95.6	153.0	Logical operation-added transfer of 640 x 480 dot data shrinked by a factor 13/16.
Arbitrary angle rotation enlarged/shrinked copy	35.2	-	71.6	103.1	Logical operation-added transfer of 200 x 200 dot data shrinked by a factor 14/16 and rotated by $\pi/8$ with relation to X and Y axes.

	1 plane → 1 plane	1 plane → 3 planes	Drawing contents
Character font normal expansion	91.5	155.2	Normal expansion of 1248 characters of 24 x 24 dot configuration font.
Character font slant expansion	119.2	196.5	Slanted expansion of 1248 characters of 24 x 24 dot configuration font.

Conformity to CGI and Upgrading to Higher Speeds

The final stage of research on graphics interface standards, such as VDI or its simplified version CGI, has been progressing. Every time a new graphics controller is announced, how the controller is designed to cope with these standards is a matter of our concern. If we may come to the conclusion first, it will be long before we really see a controller which can directly interpret CGI-defined commands. Only when the standards are established and received well and the software is mature, attempts to put the controller circuits into an LSI will start. Some controllers already claim that they conform to CGI standards, but it is an arguable statement. A CGI interpreter can be roughly divided into a command interpreting section and a device driver section. Because the command interpreting section does not depend on controllers, a different controller can even be used as long as the CPU remains the same. And even when the CPU is different, if the software has been written in C language, no problem occurs. Any difference caused by the used controller occurs only in the device driver section. Therefore, we think that how easily this device driver can be created determines the degree of the controller's conformity to CGI.

The AGDC has, as its built-in functions, almost all of the drawing functions of the CGI (Table 5). Therefore, it is enough to just create parameters which the AGDC requires. And it is not necessary to develop the software to control the complex drawing address generation and the display memory. Only when complementing drawing functions which are not built in the AGDC, software creation work needs to be considered.

Upgrading Toward 3-Dimensional Display and Image Processing:

The AGDC did not depend on the design base built during the development of the GDC. Because the GDC, which was designed when LSI integration level was no match for that of the present, had to achieve drawing functions being then limited by hardware wired logic. Under a completely new system concept, however, all of the drawing algorithms, special hardware for faster drawing functions, a preprocessor which effectively controls them, a drawing processor, individual hardware and software had to be developed and piled up little by little from the status of almost no design base. We groped in the dark for a long time. Finally the AGDC was developed as a completely new 2-generation graphics controller. For this reason, every detail regarding an optimum controller from the viewpoints of unit designers was taken into consideration.

Presently, the maximum operating frequency is 8 MHz but this will be speeded up as the size of the LSI chip is reduced. It is the natural flow of technology that, after a certain function is established, higher functions are sought for. It is now possible to imagine a leap to 3-dimensional graphics and an expansion to image processing in the near future. In addition, it will be studied how to allot functions more effectively while making better use of the functions of dual port memory and adding more functions. Anyway, products to be developed from now will be based on the AGDC, with more functions added to it. And there will less barriers than those we experienced during the development of the AGDC.

References:

1) Oguchi, "Graphic Controller LSI for Raster Scan Type CRT Which Can Draw 1 dot in 800 ns," Nikkei Electronics, Oct. 12, 1981, no. 275, pp. 186-209.

2) Pinkham, R., Novak, M. and Guttag, K., "Video RAM Excels at Fast Graphics," Electronic Design, Aug. 18, 1983, pp. 161-171

3) Kobayashi, "Development of 256 K Bit Dual Port Memory for Frame Buffer, Which Enables Continuous Serial Output," Nikkei Electronics, Aug. 12, 1985, no. 375, pp. 211-240

4) Wientjes, B., Guttag, K. and Roskell, D., "First Graphics Processor Takes Complex Orders to Run Bit-maped Displays," Electronic Design, Jan. 23, 1986, pp. 73-81

5) Mihokawa, Ueno, Yoshida, Takeda, Maejima, Katsura, "CRT Controller That Can Designate Drawing Positions by Means of Coordinates and Has Rich Commands Like Paint and Copy," Nikkei Electronics, May 21, 1984, no. 343, pp. 221-254

6) Inaba, "Comparison of Multiwindow Display Methods of Work Stations," nikkei Electronics, July 29, 1985, no. 374, pp. 141-161

7) Oguchi, Minamino, Higuchi, "Graphics Display Controller," Transistor Technology, Jan., 1983, pp. 320-343

Abbreviations are listed in Table 5 of the next page.

Table 5 List of AGDC Commands

Commands starting with A or R indicate absolute position designation or relative position designation respectively. Command names ending with A, C, D or M mean absolute address designation, coordinate designation, drawing based on the drawing processor's coordinates and drawing with newly set coordinates respectively.

Command name	Setting parameters and operation contents
READ_DP	Transfers X and Y coordinates of the drawing processor to the preprocessor.
READ_COL	Transfers color information of the designated coordinates to the preprocessor.
DOT_D	Draws 1 dot at (X#, Y#).
A_DOT_M	Draws 1 dot at (X, Y).
R_DOT_M	Draws 1 dot at (X+DX, Y+DY).
A_LINE_M	Draws a straight line between (X, Y) and (XE, YE).
A_LINE_D	Draws a straight line between (X#, Y#) and (XE, YE).
R_LINE_M	Draws a straight line between (X, Y) and (X+DX, Y+DY).
R_LINE_D	Draws a straight line between (X#, Y#) and (X+DX, Y+DY).
A_REC	Draws a rectangle defined by (X, Y) and (XS, YS).
R_REC	Draws a rectangel defined by (X, Y) and (X+DX, Y+DY).
CRL	Draws a circle defined by the center (XC, YC) and radius DX.
ARC	Draws a circular arc defined by the center (XC, YC), radius DX, start point (XS, YS) and end point (XE, YE).
CSEC	Draws a circular sector defined by the center (XC, YC), radius DX, start point (XS, YS) and end point (XE, YE).
CSEG	Draws a circular chord defined by the center (XC, YC), radius DX, start point (XS, YS) and end point (XE, YE).
ELPS	Draws an ellipse defined by the center (XC, YC), Y-direction radius DY and ratio of squared radii, DH : DV, in X and Y directions.
EARC	Draws an elliptic arc defined by the center (XC, YC), X-direction radius DX, Y-direction radius DY, ratio of squared radii, DH : DV, in X and Y directions, start point (XS, YS) and end point (XE, YE).
ESEC	Draws an elliptic sector defined by the center (XC, YC), X-direction radius DX, Y-direction radius DY, ratio of squared radii, DH : DV, in X and Y directions, start point (XS, YS) and end point (XE, YE).

Command name	Setting parameters and operation contents
ESEG	Draws an elliptic chord defined by the center (XC, YC), X-direction radius DX, Y-direction radius DY, ratio of squared radii, DH : DV, in X and Y directions, start point (XS, YS) and end point (XE, YE).
PAINT	Paints an arbitrary closed area by means of boundary point retrieval start point (X, Y) and boundary point color designation DX. Paints an arbitrary closed area with a boundary color other than that of boundary point retrieval start point (X, Y).
A_REC_FILL_A	Paints a rectangle defined by the start absolute address EAD1, horizontal direction dot count DH and vertical direction dot count DV.
A_REC_FILL_C	Paints a rectangle defined by (X, Y) and (XS, YS).
R_REC_FILL	Paints a rectangle defined by (X, Y) and (X+DX, Y+DY).
CRL_FILL	Paints a circle defined by the center (XC, YC) and radius DX.
ELPS_FILL	Paints an ellipse defined by the center (XC, YC), Y-direction radius DY and ratio of squared radii, DH : DV, in X and Y directions.
A_TRI_FILL	Paints a triangle defined by (X, Y) (XS, YS) and (XC, YC).
A_TRA_FILL	Paints a trapezoid defined by (X, Y) (XS, Y), (YS, YE) and (XE, YE).
R_TRA_FILL	Paints a trapezoid defined by left point (X, Y) and right point (XS, Y) on the upper side, relative distance DY between the upper side and lower side, relative distance DX from X of the 3rd point and relative distance XC from XS of the 4th point.
A_COPY_AA	Area-to-area transfer defined by the transfer start word address EAD2 and dot address dAD2 of a transfer source, the transfer start word address EAD1 and dot address dAD1 of a transfer destination, horizontal direction dot count DH and vertical direction dot count DV.
A_COPY_AC	Area-to-area transfer defined by the transfer start word address EAD2 and dot address dAD2 of a transfer source, the transfer start point (X, Y) of a transfer destination, horizontal direction dot count DH and vertical direction dot count DV.
A_COPY_CA	Area-to-area transfer defined by the transfer start point (XS, YS) of a transfer source, transfer start word address EAD1 and dot address dAD1 of a transfer destination, horizontal direction dot count DH and vertical direction dot count DV.

Command name	Setting parameters and operation contents
A_COPY_CC	Area-to-area transfer defined by the transfer start point (XS, YS) of a transfer source, transfer start point (X, Y) of a transfer destination, horizontal direction dot count DH and vertical direction dot count DV.
R_COPY_CC	Area-to-area transfer defined by the transfer start point (XS, YS) of a transfer source, transfer start point (XS+XC, YS+YC) of a transfer destination, horizontal direction dot count DH and vertical direction dot count DV.
PUT_A	Transfer from the main storage to a display memory area defined by the transfer start word address EAD1 and dot address dAD1 of a transfer destination, horizontal direction dot count DH and vertical direction dot count DV.
PUT_C	Transfer from the main storage to a display memory area defined by the transfer start point (X, Y) of a transfer destination, horizontal direction dot count DH and vertical direction dot count DV.
GET_A	Transfer from a display memory area defined by the transfer start word address EAD1 and dot address dAD1 of a transfer source, horizontal direction dot count DH and vertical direction dot count DV, to the main storage.
GET_C	Transfer from a display memory area defined by the transfer start point (X, Y) of a transfer source, horizontal direction dot count DH and vertical direction dot count DV, to the main storage.

Abbreviations (in alphabetical order)

AGDC: Advanced Graphics Display Controller

bitblt: bit block transfer

CAS: column address strobe

CGI: computer graphics interface

CPU: central processing unit

FIFO: first-in first-out

GDC: Graphics Display Controller

ISSCC: International Solid State Circuits Conference

JIS: Japan Industrial Standard

LSI: large scale integrated circuit

OA: office automation

RAM: random access memory

RAS: row address strobe

ROM: read only memory

VDI: virtual device interface