Vircon32:コンソールのしくみ

資料作成日2023.01.22年Carra著

これは何だ?

この文書は、Vircon32仮想コンソールがどのように動作するかを学ぶためのクイックガイドです。これは、仕様やすべての機能を詳細にカバーすることを意図したものではありません。コンソール用のプログラムを作成したい人をサポートすることだけを目的としています。このガイドは、Vircon32をC言語でプログラムすることを目的としています。アセンブラでプログラムを作成するには、この文書の範囲を超えたコンソール操作の内部詳細を知る必要があります。



まとめ

このドキュメントの最初の部分では、コンソールの一般的な機能とその基本アーキテクチャについて説明し、2番目の部分では、コンソールの各コンポーネントの操作について説明します。

[**PART 1: KNOWING THE CONSOLE 3**](#_30j0zll)

[Introduction 3](#_1fob9te)

[Parts of the console 4](#_3znysh7)

[Console architecture 8](#_2et92p0)

[**PART 2: CONSOLE COMPONENTS 12**](#_tyjcwt)

[Processor (CPU) 12](#_3dy6vkm)

[Timer 14](#_1t3h5sf)

[Input controller 15](#_4d34og8)

[Graphics chip (GPU) 17](#_2s8eyo1)

[Sound chip (SPU) 26](#_17dp8vu)

[RAM memory 30](#_3rdcrjn)

[The BIOS 31](#_26in1rg)

[Cartridge controller 31](#_lnxbz9)

[Memory card controller 31](#_35nkun2)

[Random number generator 33](#_1ksv4uv)

# パート1:コンソールを理解する

## はじめに

Vircon32は、32ビットの仮想コンソールです。これは、既存のコンソールをベースにしていない、構成されたマシンです。設計文書とそれに準拠した実装は、コンソールです。

プロジェクト全体では、他のいくつかの領域もカバーしている。エミュレータとそのゲームとともに、開発ツール、このガイドのようなガイド、テストプログラムなども存在する。

### Vircon32はいかがですか?

Vircon32は、可能な限りシンプルに設計されています。実際のマシンとしてモデル化されていますが(つまり、プロセッサ、グラフィックスチップ、コントローラ、バスなどを備えています)、そのコンポーネントは、ゲームのプログラミングや、エミュレータやその他のツールの作成を容易にするために簡素化されています。

このマシンを設計する際の2つ目の目標は、単純な技術デモやミニゲームを超える精巧なゲームの作成を可能にすることでした。このコンソールは32ビット世代のコンソールをベースにしており、その機能はその世代に沿っています(ただし、3D機能はありません)。したがって、今日ではあまりにも制限されすぎている典型的な「8ビットスタイル」を超えることができます。

## コンソールの各部

Vircon32はコンソールだけをモデル化するのではなく、ゲームパッド、ディスプレイ、カートリッジなどの要素も設計に含める必要があり、実装ではそれらを考慮に入れる必要がある。

これは何を意味するのでしょうか?例えばコントローラーです。プレーヤーは実際にはVircon32のコントローラーとは異なる種類のコントローラーでプレイしているかもしれませんし、キーボードでプレイしているかもしれません。しかし、ゲームをプログラミングする人にとって、プレイされているコントローラーは常にVircon32のゲームパッドです。実際のコントローラー入力を適応させるのはエミュレーターのタスクであり、プレーヤーはボタンをVircon32のコントローラーレイアウトに似た方法でマッピングしようとすることに興味を持つでしょう(ゲームのコントロールではその配置が当然のことと考えられるため)。

Vircon32システム自体は、古典的な32ビットコンソールとあまり違いはありません。コンソール以外には、最大4つのゲームパッド、カートリッジ、オプションでメモリカードがあります。コンソールの出力は、ある種のビデオおよびオーディオディスプレイです。

### コンソール

コンソールはゲームを実行するためのコアコンポーネントですが、その内部の仕組みについては後のセクションで説明します。外部接続レベルでは、カートリッジ入力、4つのゲームパッドポート、メモリカード用のスロット、およびディスプレイへの接続があります。

コンソールからデバイスを接続または削除するだけでなく、電源スイッチとリセットボタンを使用してデバイスの動作を制御できます。

### ゲームパッド

Vircon32のゲームパッドには、Dパッド、4つのフロントボタン、2つのショルダーボタン、そしてスタート用の中央ボタンがある。これらの要素のレイアウトは、次の図のようになります。

ゲームパッドのすべてのコントロールはデジタルです。押されているかどうかを識別するだけです。dパッドは傾斜機構を使用しているため、各軸で2つの反対方向を同時に押すことはできません。

### 画面

Vircon32の画面は、16:9のアスペクト比で640x360ピクセルの解像度を持ち、60フレーム/秒で動作します。そのため、最新のディスプレイに関しては、360pの解像度です。この解像度により、Vircon32の画像出力を、現在使用されている最も一般的な画面(720p、1080p、1440p、4K)に適応させることができ、整数スケーリングを使用して画像の変形を防ぐことができます。

色深度に関しては、画面はトゥルーカラー(ピクセルあたり24ビットのRGBカラー)を表示できます。

### カートリッジ

Vircon32カートリッジは、互いに独立した3つの読み取り専用メモリ(ROM)で構成されています。カートリッジが挿入されると、これらのメモリはそれぞれコンソール内のコンポーネントに接続されます。

* **プログラムROM:**実行されるプログラムと、そのプログラムに必要なすべてのデータが含まれます。それはメモリバスに接続され、そこからCPUはその内容にアクセスできます。
* **ビデオROM:**ゲームで表示できるすべての画像のコレクションが含まれます。グラフィックスチップ(CPU)に直接接続されます。
* **オーディオROM:**ゲームで再生できるすべてのサウンドのコレクションが含まれます。サウンドチップ(SPU)に直接接続されます。

コンソールは、カートリッジからこれら3種類のコンテンツを即時アクセスで読み取ることができるため、ロード時間が存在せず、メモリに何もコピーする必要がありません。これが可能なのは、簡素化のために、含まれるすべての画像と音声が圧縮されずに保存されるためです。

ただし、圧縮がないため、ゲームはより多くの領域を使用できる必要があります。これが精巧なゲームの作成を妨げないようにするために、カートリッジの制限が寛大に定義されています。オーディオおよびビデオROMはそれぞれ最大1 GBに達することができますが、プログラムROMは512 MBの制限があります。したがって、Vircon32カートリッジは合計で2.5 GBまで格納できます。

### メモリカード

カートリッジにはROMしか含まれていないため(つまり、読み取り専用)、コンソールではメモリカードを恒久的なストレージとして使用できます。これにより、ゲームを保存できる、より長く複雑なゲームの作成をサポートできます。これらのメモリカードの容量は1 MBです。

## コンソール・アーキテクチャ

コンソールに接続できるすべてのデバイスの基本についてはすでに説明しました。次に、コンソール自体の詳細について説明します。前のシステム全体の接続図を使用して、コンソール内に何があるかを確認するように拡張できます。基本的なレベルでは、次のようになります。

Cでゲームをプログラミングするために、これらの詳細をすべて知る必要はありません。最も重要なことは、CPUがゲームを実行するメインチップであることを知ることです。CPUは、残りのチップと通信して、特定の機能を実行するように要求できます。たとえば、ゲームのプログラムが画面に描画する関数を呼び出すと、CPUはグラフィックチップにコマンドを送信して、何をどこに描画するかを示します。

### コンソール内部コンポーネント

コンソールコンポーネントを理解するために、各コンポーネントとその機能についての一般的な概念を説明します。

* **プロセッサー(CPU):**それはメインチップであり、ゲームを実行するチップです。プロセッサは、プログラムの命令を1つずつ実行し、必要に応じて他のチップと相互作用します。
* **タイマー:**各CPU実行サイクルがいつ発生する必要があるかを制御します。また、毎秒60回、新しいフレームを開始します。これにより、画面の更新がトリガーされますが、他のチップの機能の一部も制御されます。
* **入力コントローラ:**各時点でどのゲームパッドが接続されているかを検出し、それらのDパッドとボタンの状態を問い合わせることができる。
* **グラフィックスチップ(GPU):**このチップは、カートリッジのvidro ROMにあるイメージにアクセスし、それらを使用して画面に描画します。回転やスケーリングなどのエフェクトを適用できます。
* **サウンドチップ(SPU):**SPUは、カートリッジのオーディオROM内のサウンドに直接アクセスできます。同時に16個まで再生でき、速度変更やループなどのエフェクトを適用できます。
* **カートリッジコントローラ:**カートリッジが接続されたことを検出し、その内容に関する基本的な情報を提供します。これにより、CPUはカートリッジ内のプログラムメモリを読み取ることができます。
* **メモリカードコントローラ:**メモリカードが接続されているかどうかを確認できます。CPUがカードのメモリにアクセスできるようにします。
* **乱数発生器:**これは、要求するたびに疑似乱数を生成するアルゴリズムを使用して、ランダム性をシミュレートできるようにします。
* **RAMメモリ:**これは、CPUで実行されるプログラムの作業メモリです。
* **BIOSは次のように動作します。**これはカートリッジ(プログラム、ビデオ、オーディオ)に似た内部ソフトウェアで、コンソールの起動を制御し、起こりうるハードウェアエラーに対応します。また、画面上での書き込みを可能にするテキストフォントも含まれています。

このドキュメントのパート2では、すべてのコンソールコンポーネントを詳細に見て、それらがどのように機能するかを理解し、私たちのプログラムが何をできるかを理解します。

### コンポーネント間の通信

上記で説明したコンソールコンポーネントは、個別に機能することはできません。これらのコンポーネントは、対話してコンソールを動作させるために通信できる必要があります。このため、異なる目的に使用される2つの通信チャネルまたはバスがあります。

Vircon32バスは、マスタ-スレーブ方式を使用します。この方式では、通信を制御し、コマンドを送信する独自のマスタデバイスがあります。接続された残りのコンポーネント(スレーブ)は受動的に動作し、受信した要求に応答するように制限されます。どちらのバスでも、CPUは通信を制御するデバイス(マスタ)であり、バスに接続された他のすべてのコンポーネントの使命はプロセッサにサービスを提供することです。

メモリバス

これは、メモリ(RAMまたはROMメモリ)を持つすべてのデバイスが接続するバスです。このバスに接続することによって、そのメモリ空間はアドレス範囲を受け取り、その内容はCPUによってアクセス可能になります。

CPUは、マスターとして、特定のメモリアドレスで読み取りまたは書き込み操作を実行するようにバスに要求できます。操作が実行されると、CPUは結果を含む応答を受信します。操作が実行できない場合は、ハードウェアエラーが発生し、プログラムが停止します。これは、そのメモリアドレスが存在しないか、そのアドレスでの操作が有効でないために発生する可能性があります(たとえば、読み取り専用のカートリッジメモリに書き込みが試行されます)。

コントロールバス

このバスにより、プロセッサは他のチップの機能を制御し、アクションを実行するためのコマンドを送信できます。制御メカニズムは次のとおりです:制御可能な各チップは、値の読み取りおよび/または書き込みが可能な入出力ポートのセットを外部に公開します。制御バスに接続されている場合、各チップにはポート番号の範囲が割り当てられ、CPUはそれらからデータを送受信できます。

すべてのポートで、送信または受信できる値は単一の32ビットワードです。これは、バスレベルでは、ポートの処理はメモリアドレスの処理と非常によく似ていることを意味します。また、アドレスバスと同様に、CPUが実行できない操作を要求した場合、プログラムはハードウェアエラーによって停止します。

ただし、これらの各ポートの動作は、特定のケースごとに大きく異なる場合があります。読み取り/書き込みポート、読み取り専用ポート、および書き込み専用ポートがあります。ポートの中には、単純なレジスタとして動作するものもあれば、ポートでの読み取りまたは書き込み操作が他のアクションをトリガーできるものもあります。

# パート2:コンソールコンポーネント

この2番目のパートでは、コンソールの各コンポーネントについて、その機能と使用方法について詳しく説明します。ただし、その前に、コンソール全体に影響を与える重要な詳細について説明します。CPUとメモリを簡素化するために、Vircon32のデータは32ビット単位でしか処理できません。これは、このコンソールの最小単位がバイトではなく、32ビットワードであることを意味します。プログラムを作成するときには、これはいくつかの異なる方法で影響を与えます。

* CPUレジスタは常に全体として使用されます。他のCPUのように8ビットまたは16ビットのみを処理する命令の変形はありません。
* メモリアドレスとそのオフセットは、常にバイトではなくワードで表されます。
* 同じことがデータサイズにも当てはまります。Vircon32では個々のバイトを持つことができないからです。

しかし、これは私たちにとってある種の利点を意味することもあります。他のシステムをプログラミングするときには、メモリ内のバイトのアラインメントを考慮する必要があり、バイトがビッグエンディアンまたはリトルエンディアンシステムでグループ化されている場合は、Vircon32ではこの影響を受けません。

## プロセッサ(CPU)

CPUはプログラムを実行するので、コンソールの中心的なコンポーネントです。Vircon32でゲームを作成するには、アセンブリコードを作成するのでない限り、CPU自体について多くのことを知る必要はありません。したがって、ここでは一般的な詳細についてのみ説明します。

### CPU制御

Cのプログラムから、いくつかの関数を使用してプログラムの実行を制御できます。終了フレーム()次のフレームが始まるまでCPUを一時停止します。これにより、ゲームの速度を制御できます。

| //ゲームのメインループ  一方(の真)を  {を  //現在のフレームを処理  //(.)    //次のイベントを待機  終了フレーム();  }をクリックし |
| --- |

逆に、ゲームが終了してプログラムを完全に停止したい場合は、2つの選択肢があります。1つ目はメイン関数を終了することです。2つ目は関数を使用することです。関数exit()は、コード内の任意のポイントで呼び出すことができます。

| 空所ShowGameEnding()  {を  //終了シーケンスを表示  //(.)    //ゲームが終了したので、停止します。  終了();  }をクリックし |
| --- |

### パフォーマンス

CPUは15 MHzで動作します。簡単にするために、このCPUは常に1クロックサイクルで任意の命令を実行します。Vircon32では、ゲームの多くの側面が各フレームで処理され、典型的な60 fpsで実行することは、フレームごとに最大250,000命令を実行できることを意味します。

理論的には、ゲームはより多くの処理能力を使用し、すべてのフレームではなく、たとえば2フレームごと(30 fps)に更新するように実行を制御することもできます。ただし、画面の更新やコントローラの更新などのその他の機能はプログラムから独立しており、60 fpsで機能し続けます。

### ハードウェアエラー

プログラムの実行中に、ゼロ除算などのハードウェアエラーが発生する場合があります。いずれの場合も、プログラムが停止し、コンソールに次のようなエラーメッセージが表示されます。

## タイマー

タイマーは経過時間を追跡し、他のコンポーネントが何らかのアクションを実行する必要がある場合には、適切な信号を送信します。時間の測定は、次の3つのレベルで行われます。

### サイクルカウンタ

このカウンタは、フレームの開始以降に経過したCPUサイクルを測定します。つまり、各フレームは0～249999でカウントされます。これをCからクエリーするにはget\_cycle\_counter()[get\_cycle\_counter()]関数を参照してください。ゲームでこれを使用するのは一般的ではありませんが、いくつかの用途があります。たとえば、フレームの終わりの前の値を読み取ってCPU負荷を内部的に把握し、デバッグモードで画面に表示できます。

| //使用されているCPUの割合を計算する  内部UsedCycles=get\_cycle\_counter();  浮くCPUPercentage=100.0\*UsedCycles/250000;  //次のフレームを待機  終了フレーム(); |
| --- |

ただし、サイクルカウンタを使用してイベントの時間を測定することはできないことに注意する必要があります。Vircon32エミュレータは、各フレームの開始が必要なときに発生することを保証するために必要です(一定の間隔で、1秒あたり60回)。ただし、そのフレームのサイクルを適切と思われる任意のレートでエミュレートすることは自由です。プログラムが125,000に等しいサイクルカウンタを読み取ったとしても、正確に1/120秒がリアルタイムで経過したと考えるべきではありません。

### フレームカウンタ

フレームカウンタは、コンソールがオンになったときに0から始まり、その瞬間から経過したフレームの数を測定します。このカウンタは、時間を測定する主要な方法であるため、プログラムで最も一般的に使用されるカウンタです。この場合、関数get\_frame\_counter()[get\_frame\_counter()]が使用されます。たとえば、メインループでは次のように実行できます。

| //最初のイベントで現在の時間を保存  もし(イベント発生)  StartingFrames=get\_frame\_counter();    //(.)  //1秒後にそのイベントへの応答をアクティブにする  もし(イベント発生)  (の場合get\_frame\_counter()-StartingFrames>=60)  TriggerResponse(&イベント);    //(.)  //次のフレームを待機  終了フレーム(); |
| --- |

さらにsleep()[スリープ]関数を使用します。この関数は、フレームカウンタをリレーして特定の時間待機します。これは、たとえば、小さな待機を含めたいシーントランジションで役立ちます。

| //シーンを黒にフェード  FadeToBlack(&シーン1);  //画面を0.5秒間黒く保つ  スリープ(30)  //次のシーンにトランジションを作成する  FadeFromBlack(&シーン2); |
| --- |

### 日付と時刻

タイマーは、コンソール自体に関連する時間を制御するだけでなく、現在の日付と時刻を示す内部クロックも備えています。これらは、次の方法で照会できます。get\_date()[get\_date()]およびget\_time()[get\_time()]これらの関数は、ネイティブのVircon32バイナリ形式(それぞれint)で情報を提供しますが、次のコードを使用して通常の人間の形式に変換できます。

| //現在の日付と時刻をVirconのネイティブ形式で読み込む  内部VirconTime=get\_time();  内部VirconDate=get\_date();    //日付を人間の形式に変換する  //(年、月、日の構造)  date\_info HumanDate;  translate\_date(VirconDate、HumanDate);  //時間を人間の形式に変換する  //(時、分、秒の構造)  time\_info HumanTime;  translate\_time(VirconTime、HumanTime); |
| --- |

ゲームは、これらの値を使用して、ゲームがメモリカードに保存された日時を格納し、ゲームがロードされたときにそれらをプレーヤに示すことができる。

## 入力コントローラ

このコントローラーは、コンソールの4つのポートのそれぞれに接続されたゲームパッドの状態を常に監視します。Dパッドとボタンに関する情報を提供するだけでなく、どのゲームパッドが接続されているかも教えてくれます(ゲームパッドはいつでも接続および切断できます)。

知っておくべき重要な概念は、4つのポート(ゲームパッドが接続されているかどうかにかかわらず)がありますが、ゲームパッドコントローラで選択されているのは常にそのうちの1つだけであるということです。4つのゲームパッドは常に最新の状態に保たれますが、クエリは常に現在選択されているゲームパッドで行われます。既定で選択されているゲームパッドは最初のゲームパッドです。

このプログラムではselect\_gamepad()[選択ゲームパッド]選択したゲームパッドを変更します。ゲームパッドを選択すると、gamepad\_xxx関数を使用して、何が押されているかを確認できます。これは、読みたい方向やボタンによって異なります。

| //プレイヤー2のゲームパッドの状態を読み取ります  select\_gamepad(1);<文字列>  //ボタンAを押したときに撮影  もし(gamepad\_button\_a()==1)と入力します。  シュート(&Player2);  //現在左が押されている場合は、後ろに進みます  もし(gamepad\_left()>0)を返します。  プレイヤー2.x-=3; |
| --- |

Vircon32ゲームパッド自体では、ボタンと方向の状態は単純な「はい」または「いいえ」(押された、または離された)です。しかし、コードからわかるように、クエリ関数はブール値ではなく整数を返します。これは、ゲームパッドコントローラがより多くの情報を提供するためです。コントロールが押されたかどうかを知るために使用できるのは符号だけですが、値自体はコントロールが現在の状態にあった時間を示します。

そのため、この例では、値が1の場合にのみ撮影します。ボタンがこのフレームで押された直後です。押されたフレームごとに発射を続けることはありません。一方、方向を見るときは、時間には関心がなく、その符号をチェックするだけです。これは、フレームごとにプレーヤーを移動するためです。

ボタンや方向から受け取る状態には常に符号が付いています。ゲームパッドのコントローラは、それが決してゼロにならないことを保証します。これにより、プログラミングが容易になります。

### ゲームパッドの状態の更新

コントローラはイベントを使用して動作しませんが、各フレームの最初に、すべてのゲームパッドのプレスを自動的にチェックし、それらの状態を更新します。プログラムはこのために何もする必要はありません。必要なときにクエリを作成するだけです。

## グラフィックスチップ(GPU)

GPUは、プロセッサとは別に、Vircon32の最も複雑なコンポーネントです。ただし、非常にシンプルなグラフィックスチップです。実際には、プロセッサが要求した場合にGPUが実行できるコマンドは2種類しかありません。

* 一定の色で画面をクリアすることができます。
* カートリッジ内のテクスチャの一部から領域を画面上に描画できます。

### パラメータのないコマンド

GPUが実行できる描画コマンドには、いくつかのパラメータが必要です。たとえば、テクスチャの一部を画面上に描画する場合、どの領域について説明しているのか、画面上のどこに配置するのかを知る必要があります。これらのコマンドを関数として使用する場合、プロセッサは次のような方法でパラメータを渡すことによってコマンドを呼び出します。

DrawRegion(領域,X,Y)

ただし、GPUはこれらのコマンドをポートの1つで単純な値として受信するため、コマンドはパラメータを受信できません。GPUでは、コマンドの実行時に使用される一連の設定可能なパラメータ(後述)が内部的に存在します。プログラムは、コマンドを送信する前にこれらの値を設定する必要があります。したがって、上記の関数には、実際には次のフェーズがあります。

SelectedRegion=リージョン

DrawingPointX=X

DrawingPointY=Y

描画領域()

場合によっては、標準Cライブラリは作業を簡単にしようとし、GPU(または他のチップ)からのコマンドを関数であるかのように使用できるようにします。たとえば、画面の中央に領域を描画するには、ライブラリでは次の2つの選択肢が可能です:

| //オプション1:いくつかのパラメータがすでにコマンドに統合されています  select\_region(RegionCharacter);選択領域  draw\_region\_at(320,240);(描画領域)  //option 2:すべてのパラメータを個別に指定したコマンド  select\_region(RegionCharacter);選択領域  set\_drawing\_point(320,240);図面の点を設定  参照):draw\_region(); |
| --- |

### 画面のクリア

このコマンドは非常に簡単です。クリアカラーという1つの内部パラメータにのみ依存します。このコマンドは、このカラーを画面全体に均一に適用します。

色はRGBA形式の32ビット値です。いくつかの定義済みの色はすでにCライブラリに存在します。例えば、画面を青色で消去するには、次のようにします。

| clear\_screen(カラー\_青); |
| --- |

事前に定義された色を使用することに加えて、それらの数値を書くこともできます。しかし、その場合、Vircon32は各単語のバイトをリトルエンディアンシステムで保存するので、その記述表現は反転されている(ABGR)ことを考慮する必要があります。例えば、プログラムでは、青色はFFFF0000(0000 FFFFFFではなく)と書かれます。より簡単なオプションは、Cライブラリの関数を使用して色を作成することです:

| //アルファを使用する場合は、make\_color\_rgbaも使用できます。  clear\_screen(make\_color\_rgb(0,0,255));クリアスクリーン |
| --- |

画面をクリアするときに、アルファチャンネル(不透明度)を使用して部分的なクリアを行うことができます。アルファ=128の場合に黒でクリアを行うと、画面イメージは暗くなりますが、完全に黒くはなりません。

### GPUテクスチャ

2つ目のタイプのGPUコマンドを説明するには、テクスチャの概念を理解する必要があります。ご存知のように、GPUはカートリッジに格納された一連のイメージを処理します。GPUは1024x1024ピクセルの標準サイズのイメージのみを処理できますが、カートリッジ上の各イメージは(その制限まで)異なるサイズを持つことができます。

これを補正するために、カートリッジがGPUに接続されると、カートリッジイメージはビデオメモリに格納され、このイメージに示されているように配置されます。両方とも同じ最初のピクセル(0,0)を持つため、すべてのピクセルの座標が保持されます。これらのイメージはすでに格納され、GPUに適合されており、テクスチャと呼ばれます。

### テクスチャリスト

GPUには、カートリッジ上の最大256個のテクスチャに加えて、BIOS用の1個の追加テクスチャに十分なスペースがあります。これらのテクスチャはすべて番号付きリストとして処理されます。BIOSテクスチャ(常に存在します)には特別なID-1が与えられます。カートリッジからのテクスチャには、0～255のIDで番号が付けられます。

番号付けは、カートリッジ内のイメージと同じ順序に従います。最初に存在するイメージはID 0、2番目はID 1のようになります。カートリッジで使用されていないID(最大255)は、関連付けられたテクスチャなしで残され、プログラムはそれらを使用できません。

存在するすべてのテクスチャの中で、コマンドは常にGPUで現在選択されているテクスチャに適用されます。デフォルトでは、選択されたテクスチャはBIOSテクスチャです。これは、常に存在することが保証されている唯一のテクスチャであるためです(カートリッジにテクスチャが含まれていない場合があります)。

私たちのプログラムではselect\_texture()[選択テクスチャ](select\_アイコンの横にある矢印をクリックします。プログラムに複数のテクスチャがある場合は、通常、テクスチャに名前を付けることをお勧めします。

| #define TextureLevels 0  #TextureMenus 1を定義  //(.)  select\_texture(TextureLevels); |
| --- |

### 各テクスチャの領域

GPUによって使用されるテクスチャは、画面自体よりも大きくなります。このため、GPUはテクスチャを直接描画するのではなく、そのテクスチャの領域のみを描画します。通常、ゲームの複数のイメージは、次の例のように同じテクスチャにまとめて保存されます。

領域を区切る境界は、コンソールでは実際には必要ありません。しかし、テクスチャを作成して使用するときに、プログラマにとっては簡単になります。

各GPUテクスチャには、0～4095のIDを持つ4096の領域があります。領域を定義する方法は、次の図に示すように、テクスチャの3つのポイントを使用することです。

最初の2つの点を使用して、描画する領域の長方形を定義します。3番目の点は、画面にイメージを配置するための参照として使用されます。たとえば、画面の中央(点320,180)に描画する場合、領域の参照点はこれらの座標に一致するように作成されます。

選択されたテクスチャIDが常に存在するように、GPUは選択された領域IDも保存します。この領域IDはコマンドで使用されます。選択された領域を変更するにはselect\_region()[選択領域]ただし、選択された領域IDはGPUのグローバルパラメータであることに注意してください。個々のテクスチャごとに異なる選択領域IDが保存されることはありません。

C関数を使用すると、いくつかの方法で領域を定義できますが、最も基本的な方法は次のとおりです。領域定義()これにより、リージョンを定義する3つのポイントが適用されます。

| #define TextureLevels 0  #define RegionPlatform 50  //プラットフォーム領域を定義する  select\_texture(TextureLevels);  select\_region(RegionPlatform);  define\_region(0,0,324,49,112,49);定義領域//下部の中心点を参照 |
| --- |

ここでわかるように、テクスチャのIDとは異なり、使用する領域IDを自由に選択できます。各テクスチャには常に4096の領域が存在し、それらを設定するだけです。ID=0から開始する必要はなく、IDが連続している必要もありません。

### リージョンを描画するコマンド

すでに説明したように、GPUが画面上に領域を描画するときは、常に内部パラメータに従って描画されます。つまり、現在選択されているテクスチャと領域が使用され、イメージを配置するために、現在の描画ポイントである他の2つのパラメータ(座標x、y)も使用されます。前の例のコードでは、次のように変更できます。set\_drawing\_point()[図面ポイントの設定]、または既に統合されている関数を使用します。

ただし、テクスチャ内の領域を画面上に描画できるだけではありません。GPUは、領域に変換を適用することもできます。この場合は、回転やスケーリングを適用します。領域を描画するときに、これら2つの変換をそれぞれ適用するかどうかを選択できます。このため、合計4つのコマンドがあります。この表では、それぞれの場合にどのC関数が使用されるかを確認できます。

| 機能 | スケーリング | 回転 |
| --- | --- | --- |
| draw\_region()[領域描画] | いいえ | いいえ |
| draw\_region\_zoomed()[描画領域ズーム] | はい | いいえ |
| draw\_region\_rotated()[描画領域の回転] | いいえ | はい |
| draw\_region\_region()(描画領域rotozoomed) | はい | はい |

変換を適用しないコマンドを使用すると、GPUはそのコマンドに関連付けられたパラメータを無視します。

### 拡大/縮小して描画

スケーリングを適用する領域を描画する場合、このコマンドでは、XおよびYスケーリング係数の2つの追加パラメータを使用します。これらの係数は、互いに異なるだけでなく、負の値にすることもできます。負のスケーリングの結果、イメージはこの軸に沿って反転されます。たとえば、次のように記述できます。

| //XとYでスケーリングが異なる  select\_region(RegionPlatform);  set\_drawing\_scale(0.7,3);図面尺度の設定  draw\_region\_zoomed\_at(320,180);(描画領域のズーム倍率) |
| --- |

GPUによって表示される結果は次のようになります。

### 回転して描画

回転を使用して領域を描画する場合、コマンドでは描画角度という別のパラメータが使用されます。Vircon32の角度は常にラジアンで測定されます(180度はπラジアンに相当します)。また、画面座標は一番上から始まるため、Y軸が反転されます。これにより、角度は反時計回りではなく時計回りに大きくなります。例を次に示します。

| //反時計回りに回転するには角度が負です  select\_region(RegionPlatform);  set\_drawing\_angle(-pi/6);図面角度設定(-pi/6)//pi=180°,luego-pi/6=-30°。  draw\_region\_rotated\_at(320,180);回転された領域の描画 |
| --- |

その場合、画面には次のように表示されます。回転の中心は常に領域の参照ポイントです。つまり、変換を行っても、参照ポイントは常にGPU描画ポイントと一致するように描画されます。

### カラーの変更

領域を描画する場合は、カラーに変換を適用することもできます。GPUには「乗算カラー」と呼ばれるパラメータがあり、RGBAコンポーネントに乗算効果を適用して描画されるカラーを修正します。

OpenGLを知っている場合、効果はglColorと同じです。つまり、中間色は白(デフォルトの乗算カラー)で、乗算カラーのRGBコンポーネントが暗くなると、描画される領域のコンポーネントが暗くなり、フィルタとして機能します。アルファコンポーネントも乗算されるため、さまざまな不透明度レベルで領域を描画できます。

たとえば、次のコードを記述して、領域の色から赤のコンポーネントを削除できます。

| //カラーを緑と青で乗算し、赤を乗算しない(この場合はシアンカラー)  select\_region(RegionPlatform);  set\_multiply\_color(make\_color\_rgb(0,255,255));設定乗算カラー  draw\_region\_at(320,180);(描画領域) |
| --- |

視覚的には、このコードで画面に表示される結果は次のようになります。

乗算カラーは、thatdraw地域のすべてのコマンド(4つのバリエーションすべて)に適用されますが、clear screenコマンドには影響しません。

### カラーブレンディングモード

カラーブレンドは、背景にイメージを描画する方法です。ブレンドを変更することによって得られる効果を理解するための最も簡単な方法は、Photoshopなどの描画プログラムでさまざまな種類のレイヤーを表示することです。

Vircon32のGPUには、3種類のカラーブレンディングモードが用意されています。

* **アルファモード:**これはデフォルトの描画モードであり、直感的な方法で領域を描画します。カラーは変更されず、領域のアルファコンポーネントを不透明度として使用します。その結果、描画する背景とブレンドする半透明のイメージを作成できます。
* **追加モード:**描画プログラムでは「Linear Dodge」とも呼ばれます。領域の色が背景の色に追加され、照明効果が作成されます。このモードでは、黒が中間色です。
* **減算モード:**描画プログラムでは「差分」とも呼ばれます。描画領域の色が背景の色から差し引かれ、遮光効果が可能になります。これにより、この場合は黒も中間色になります。

ブレンドモードを変更するにはset\_blending\_mode()[ブレンドモード設定]上の図に示すモード名を使用します。

異なるモードを組み合わせると、次のような環境効果が得られます。

色の変更とは異なり、異なる描画モードは、領域を描画するときには適用されませんが、画面をクリアするときにも適用されます。

### GPUパフォーマンス

決められたパフォーマンス制限がなければ、グラフィックスチップは適切に定義されません。このGPUのパフォーマンスは、各フレームに描画されるピクセル数に基づいています。

各フレームでGPUは、フルスクリーンの9倍(1080pの解像度で1つのフルスクリーンに相当)に等しい残りのピクセルのカウンタから開始します。コマンドが実行されるたびに、GPUは消費されるピクセル数の非常に基本的な概算を計算します。GPUが現在のフレームで十分な容量を持っていない場合は、次のフレームまでコマンドを実行する要求を無視します。

各コマンドによって消費されるピクセルの計算は、非常に正確ではない場合でも、迅速かつ単純であることが意図されています。ゲームを作成するためにこのプロセスを知る必要はありませんが、GPUを最大限に活用するために必要な場合があります。プロセスは次のとおりです:

* の「有効幅」が計算されます。これは、描画されるオブジェクトの幅を画面ピクセル単位で表したもので、該当する場合はX方向のスケーリング(符号なし)が適用されます。画面幅よりも大きい場合は、640ピクセルに制限されます。
* の「有効高さ」は、前のポイントと同様に計算されます。画面の高さよりも大きい場合は、360ピクセルに制限されます。
* 消費されるピクセルは、「有効幅」x「有効高さ」として計算されます。
* すべての操作が同じようにコストがかかるわけではないため、実行される操作に応じて、消費されるピクセルは次の要因によって変更されます。
  + 画面をクリアする場合:-50%
  + 尺度を適用して領域を描画する場合:+15%
  + 回転を適用して領域を描画する場合:+25%  
    (両方の効果が適用される場合、係数は+40%です)

図からわかるように、この計算では描画位置が無視されます。これは、領域のうち画面の外側にある部分も使用ピクセルとしてカウントされることを意味します。また、現在の描画モードはパフォーマンスに影響を与えません。それらはすべて同じくらい高価だと考えられています。

## サウンドチップ(SPU)

サウンドチップにはGPUと共通の機能がいくつかありますが、よりシンプルです。GPUと同様に、カートリッジに対して最大1024のサウンドの番号付きリスト(IDは同じ順序で0～1023)と、BIOSサウンド用に予約されたID=-1があります。

このサウンドのリストには、常に選択されたサウンドがありselect\_sound()[サウンド選択]対応するIDを指定します。選択されているデフォルトのサウンドは、常に存在する唯一のサウンドであるため、BIOSサウンドです(カートリッジにはサウンドがない場合があります)。

グラフィックチップとの違いは、サウンドがそのまま使用されることです。サウンドは標準サイズである必要はなく、テクスチャ領域で発生したようなサウンドの一部を使用して再生することはありません。多くの場合、サウンドで何かを設定する必要はありません。

### 再生チャンネル

サウンドチップには、サウンドを再生できる16のチャネルがあります。これらのチャネルには、0～15のID番号が付けられています。SPUでは常にチャネルが選択されています(デフォルトではID 0)。このチャネルはselect\_channel()[チャネル選択]チャンネルを選択すると、リストから(そのIDを使用して)サウンドを割り当て、再生を開始できます。

各サウンドチャネルは互いに独立しており、すべてのサウンドを同時に再生できます。ただし、多くのサウンドが同時に再生されている場合は、再生音量を下げない限り、スピーカー信号が飽和状態になることに注意する必要があります。

サウンドチャネルは、再生中、一時停止中、停止中の3つの状態のいずれかになります。チャネルにサウンドが割り当てられていない場合、またはチャネルの再生が終了した場合、チャネルの状態は常に停止します。チャネルの状態は、次の方法で問い合わせることができます。get\_channel\_state()[チャネル状態の取得](該当日本語なし

| //サウンドの再生が終了するまで待機する  一方(get\_channel\_state(ChannelSoundEffects)!=チャネル停止)  終了フレーム(); |
| --- |

### サウンドコマンド

Vircon32 SPUには6つのサウンドコマンドがあります。一方では、再生、一時停止、停止のコマンドを使用して特定のチャネルを操作できます。他の3つのコマンドを使用すると、SPUではすべてのチャネルを同時に停止、一時停止、再開できます。

Cからサウンドを再生するオプションは複数あります。play\_sound()[サウンド再生]無料のチャンネルを検索させます。ただし、場合によっては(たとえば、BGMの場合)、再生する特定のチャンネルを選択する必要があります。コードでは、次のように2つの方法が使用されます。

| //チャンネル15をBGM用に予約  play\_sound\_in\_channel(レベル1音楽、15);  //他のチャンネルでサウンドエフェクトを再生する  //(ただし、ボリュームを変更するには、どのボリュームにするかを指定する必要があります)  内部UsedChannel=play\_sound(レベル1音楽);  set\_channel\_volume(UsedChannel、0.25年) |
| --- |

すべてのチャンネルに影響するコマンドは、ゲームのコンテキストが変化する状況で役立ちます。たとえば、一時停止メニューですべてのサウンドを一時停止し、一時停止メニューを終了したときに再開するには、次のようにします。

| //一時停止メニューを入力するとき  pause\_all\_channels();全てのチャンネルを一時停止する  //メニュー自体を表示する  ShowPauseMenu x  //ゲームに戻るとき  参照):resume\_all\_channels(); |
| --- |

### 再生音量

サウンド再生チャンネルでは、音量はset\_channel\_volume()[チャンネルボリューム設定](範囲は0～10)。デフォルトのボリュームは0.5です。チャンネルのボリュームは、変更しない限り変更されません。また、異なるサウンドを再生しても同じままです。

### 再生速度

チャンネルを使って再生速度を変更するにはset\_channel\_speed()[チャネル速度設定]アイコンの横にある矢印をクリックします。別の速度で再生すると、サウンドのピッチも変わります。指定できる範囲は0～100で、既定の速度は1です。このエフェクトは、楽器の異なる音をシミュレートするのに便利です。

音量と同じように、チャンネルの速度は変更しなければ同じままです。

### 再生時のジャンプ

チャンネルは、そのサウンドのサンプルに沿って再生位置を継続的に進めることによって、割り当てられたサウンドを再生します。チャンネルでは、この再生位置をset\_channel\_position()[チャンネル位置設定]たとえば、前方にジャンプした場合、下の図のようにサウンドが再生されます。

### ループ再生

SPUチャネルを使用すると、ループ内のサウンドを再生できます。このオプションはset\_channel\_loop()[チャネルループ設定]ただし、各GPUサウンド自体にも、ループ再生するかどうかを決定するパラメータがあります。これはset\_sound\_loop()[サウンドループ設定]チャンネルがサウンドの再生を開始すると、その特定のサウンドに合わせて再生ループが自動的にオンまたはオフになります。デフォルトでは、すべてのサウンドとチャンネルのループは無効になっています。

デフォルトでは、サウンド全体に対してループが作成されますが、開始点と終了点をマークすることで部分的なループを作成できます。再生は次の図のように行われます。

これを実現するには、そのサウンド内でループする特定の領域を区切る必要があります。これは次の例のように行うことができます。

| //このサウンドをループで再生するように設定する  select\_sound(SoundCarEngine);  set\_sound\_loop(サウンドループの設定)真)  //ループ領域を設定する(位置はサンプルで指定)  set\_sound\_loop\_start(1348)を設定します。  set\_sound\_loop(12507);サウンドループの設定  //再生コマンド時に、チャネルループが自動的に有効になります  int EngineChannel=play\_sound(SoundCarEngine);  //(.)  //サウンドを終了させるためにループを無効にする  select\_channel(EngineChannel);  set\_channel\_loop()は偽) |
| --- |

## RAMメモリ

RAMメモリはパッシブコンポーネントであるため、あまり知られていません。そのサイズは16 MBですが、最小単位が32ビットワードであるため、4 MWとしてサイズを測定することがより適切です。

## BIOS

BIOSは、コンソールに常に存在する内部ソフトウェアです。コンソールが起動すると、Vircon32のロゴが表示され、カートリッジが接続されているかどうかを確認します。カートリッジが接続されている場合は、BIOSに制御が移り、接続されていない場合は、警告画面が表示されます。

ハードウェアエラーが発生した場合、BIOSはそれを処理するための制御を再び受け取り、終了すると実行を停止します。

Vircon32 BIOSには、常に1つのテクスチャと1つのサウンドが含まれており、GPUとSPUの両方でID番号-1が予約されています。このテクスチャに含まれる要素の1つは、画面上にテキストを書き込むことができる10x20ピクセルのテキストフォントです。

## カートリッジコントローラ

このコントローラを使用すると、カートリッジに保存されているコンテンツ(プログラム、テクスチャ、サウンド)にコンソールからアクセスできます。これらはすべて自動的に行われ、プログラム内でそれに関連することを行う必要はありません。

また、コントローラは、接続されているかどうか、含まれているテクスチャとサウンドの数、プログラムROMのサイズなど、カートリッジに関する基本的な情報をコンソールで問い合わせることもできます。この情報は、通常、使用する要素をすでに知っているゲームプログラムでは必要ありません。したがって、カートリッジコントローラにテストプログラムからこの情報を要求することは意味があります。

BIOSは、コンソールの起動中にカートリッジが接続されているかどうかを確認する必要があります。その後は確認する必要はありません。Vircon32では、コンソールの電源を入れるとカートリッジの入力がロックされ、最初にコンソールの電源を切らない限りカートリッジの挿入や取り外しはできません。

## メモリカードコントローラ

メモリカードは、接続されると、読み取り/書き込みメモリ領域としてCPUにアクセスできます。そのサイズは1 MBですが、最小単位は32ビットワードであるため、サイズを256 KWと測定することがより適切です。Cプログラムでは、次の関数を呼び出すことによって、コンソールにカードがあるかどうかを確認できます。カードは接続されています()接続されていないカードにアクセスしようとすると、ハードウェアエラーが発生するため、このチェックを行うことが重要です。

### 各ゲームの特徴

プログラムは他のメモリ領域と同様にメモリカードにアクセスできますが、メモリカードには(他のゲームからの)ゲームセーブが含まれている可能性があるため、カードを使用する前にカードに実際に何が含まれているかを確認することをお勧めします。

このため、一般的には、カードの最初の20語がゲームの「署名」として使用されます。これは、データを保存したのがゲームであるか(署名が予想されるものと一致する場合)、別のゲームであったかを識別する既知の値のセットです。したがって、カードをチェックするときに不明な署名が検出された場合、互換性がなく、エラーを引き起こす可能性のあるゲームをロードしようとすることを防ぎます。

署名を処理するにはカード読み取り署名(card\_read\_signature)card\_write\_signature()はおよびcard\_signature\_matches()は(次の例を参照)。

| //ゲームの署名を作成する  game\_signature GameSignature  memset(GameSignature,0,sizeof(game\_signature));<文字列>  strcpy(GameSignatureマイ・テスト・ゲーム)  //(.)  //ゲームを保存するとき  card\_write\_signature(&GameSignature);  SaveGameData x  //(.)  //ゲームをロードするとき  もし(card\_signature\_matches(&GameSignature))  LoadGameData x |
| --- |

メモリカードが作成された後にまだ使用されていない場合は、メモリカードが空である可能性もあります。プログラムは、署名にゼロしか含まれていない場合、カードが空であると想定できます。標準ライブラリには、次のものがあります。card\_is\_empty()チェックしてください。

カードが使用できることを確認したら、関数を使用してデータの読み取りと書き込みを行うことができます。カード読み取りデータ(card\_read\_data()およびカード書き込みデータ()たとえば、前の例のload関数とsave関数は次のようになります。

| //ゲーム内のデータ  game\_signature MySignature//タイプは"memcard.h"ですでに定義されています  GameStateの現在の状態;//定義された構造体  //(プログラムの後半)  ボイドSaveGameData()  {を  //署名の直後にデータが書き込まれる  内部OffsetInCard=サイズ(ゲームの署名);  card\_write\_data(&現在の状態、OffsetInCardサイズ(現在の状態));  }をクリックし  ボイドLoadGameData()  {を  //データは署名の直後に読み込まれます。  内部OffsetInCard=サイズ(ゲームの署名);  card\_read\_data(&現在の状態、OffsetInCardサイズ(現在の状態));  }をクリックし |
| --- |

## 乱数発生器

この乱数発生器は、初期の乱数(シード)からスタートし、CPUが擬似乱数を要求するたびに、その乱数から擬似乱数の系列を生成することができます。同じシードから、乱数発生器は、たとえ異なるVircon32コンソール上であっても、常に同じ系列を生成します。

シードを変更するには参照):srand()を繰り返し呼び出してシーケンスから番号を抽出します。関数rand()Cプログラムでは、現在の時間を使用してシードを作成し、生成されたシーケンスをゲームセッションごとに異なるものにするのが一般的です。

| //新しい数値のシーケンスを初期化する  srand(get\_time());のようになります。  //サイコロの振りをシミュレート:1～6  内部DiceThrow=1+rand()%6; |
| --- |