32ビット仮想コンソール

システム仕様

第2回:コンソール・アーキテクチャー

資料作成日2023.01.08年Carra著

これは何だ?

このドキュメントは、Vircon32システム仕様のパート2です。この一連のドキュメントは、Vircon32システムを定義し、その機能と動作を詳細に記述した完全な仕様を提供します。

この仕様の主な目的は、Vircon32システムとは何か、およびゲームシステムが準拠していると見なされるために実装される必要がある方法の標準を定義することです。また、Vircon32は仮想システムであるため、これらのドキュメントの重要な2番目の目的は、独自のVircon32実装を作成するための知識を誰にでも提供することです。

Vircon32について

Vircon32プロジェクトはCarraによって独自に作成されました。Vircon32システムとその関連資料(ドキュメント、ソフトウェア、ソースコード、アート、およびその他の関連要素を含む)は、元の作成者が所有しています。

Vircon32は無料のオープンソースプロジェクトで、誰でもゲーム機をプレイし、そのためのソフトウェアを開発できるようにすることを目的としている。この詳細については、使用可能な各ソフトウェアに含まれているライセンステキストを参照してください。

このドキュメントについて

このドキュメントは、Creative Commons Attribution 4.0 License(CC BY 4.0)に基づいて提供されています。ライセンスの全文は、Creative CommonsのWebサイトで読むことができます。

<https://creativecommons.org/licenses/by/4.0/>

**まとめ**

仕様のパート2では、最初にすべてのコンソール・コンポーネントとそれらに割り当てられたタスクを識別し、次に、異なるチップが通信し、相互作用して、それらが統合されてコンソールとして動作する方法を説明します。

ここでは、すべてのコンソール・コンポーネントの機能に影響を与えるメカニズムとデータ・プロパティーについても説明します。

[1 Console internals 3](#_30j0zll)

[2 Timing control 6](#_1fob9te)

[3 Communication buses 7](#_3znysh7)

[4 Memory bus 8](#_2et92p0)

[5 Control bus 9](#_tyjcwt)

[6 Chip commands 11](#_3dy6vkm)

[7 Internal data formats 11](#_1t3h5sf)

[8 Endianness 13](#_4d34og8)

[9 Hardware errors 14](#_2s8eyo1)

# 

## 1コンソール内部

第1部の導入部を読むと、Vircon32の基本的なシステムレベルのアーキテクチャが次の図に従っていることがわかります。

この図では、簡単なカラーコードを使用して、コンソールに接続できる外部コンポーネントの各タイプを識別しています。入出力用のデバイスと、外部メモリに接続するデバイスがあります。これらのデバイスは、すべてオプションであるため、コンソールに接続することも、接続しないこともできます。接続しない場合でも、コンソールは動作しますが、明らかに追加機能はありません。

第1回では、これらすべての外部デバイスの一般的な概要を説明したので、ここでは一時的にコンソールの環境を省略し、コンソール自体を見てみましょう。次の詳細レベルに進むために、前の図を拡張して、コンソールがブラックボックスではなくなるようにすることができます。この2番目の図では、内部コンポーネントを区別できます。

わかりやすくするために、この図では、これらの内部コンソールチップ(後のセクションで説明します)を通信するデータバスを省略しています。示されている「制御ロジック」ブロックに関しては、コンソール全体にまたがるメカニズムを考慮するより簡単な方法です。この場合、前記メカニズムは、コンソールのすべてのコンポーネントによって受信される制御信号であり、電源、リセット、およびタイミングに関連するイベントを送信します。

### 1.1コンソールチップ

コンソール内の主なコンポーネントは、コンソール機能の実行を担当するさまざまなチップです。その他のコンソールコンポーネントは二次的なもので、これらのチップをサポートするために存在します。Vircon32コンソールのチップは次のとおりです。

* **プロセッサー(CPU):**プロセッサは、プログラム命令を1つずつ実行し、必要に応じて他のチップと相互作用する。
* **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:**4つのゲームパッドのうちどのゲームパッドが接続されているかを検出します。これにより、プログラムはDパッドとボタンの状態を照会できます。
* **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知ることができます。存在する場合、CPUはそのカードのメモリにアクセスできます。
* **グラフィックスチップ(GPU):**このチップは、カートリッジのビデオROMにあるイメージにアクセスし、それらを使用して画面上に描画します。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。
* **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:**CPUで実行されているプログラムの作業メモリです。そのサイズは16 MB、つまり32ビットワードでは4 MWです。
* **BIOSの場合:**これは読み取り専用メモリで、構造はカートリッジに似ています。これには、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する内部ソフトウェアが含まれています。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。

### 1.2通信バス

説明されているコンソールチップは、単独では適切に機能することができません。通信および相互作用が可能である必要があります。詳細なアーキテクチャ図では、異なるチップを接続する2つの通信バスを意図的に省略しています。これらは次のとおりです。

* **メモリバス:**このバスを使用すると、さまざまなデバイスが内部メモリをCPUにアクセスできるようになります。このバスは、すべてのローカルアドレス範囲を、同じ方法で読み書きできるグローバルメモリアドレスのマップに結合します。
* **コントロールバス:**このバスの目的は、CPUが各チップによって公開された一連の「制御ポート」にアクセスできるようにすることです。これらのポートからの読み取りと書き込みによって、実行中のプログラムはこれらのチップの機能を制御できます。

これらの通信バスとその接続構成については、第3章で詳しく説明します。

### 1.3コントロール信号

すべてのコンソールチップが必要に応じてイベントに対応できるようにするために、すべてのコンソールチップで使用可能にする必要があるグローバルなコンソールレベルのイベントがいくつかあります。この目的のために、コンソールは次の制御信号を使用します。

* **電源:**これ自体は実際には信号ではありません。ハードウェア実装では、すべてのコンポーネントに電力を送信し、ボタンまたはスイッチで制御する必要があります。Vircon32エミュレータの場合、これは単純なオン/オフの方法に縮小できます。
* **リセット:**Reset信号は、ユーザが直接トリガーすることもできますが、コンソールの電源を入れるたびに発生します。これは、既知の初期状態に戻すために、すべてのコンポーネントに送信されます。
* **新しいフレーム:**この信号は、Vircon32ビデオ信号が60 fpsでリフレッシュされるたびにトリガされます。Vircon32がタイミングを処理する方法(次のセクションを参照)により、一部のチップがアクションを実行するためには、このタイミングも送信する必要があります。
* **新しいサイクル:**サイクルは、Vircon32内の最小時間単位です。サイクル信号は、CPUおよびその他のコンポーネントのクロック信号として機能することを目的としています。Vircon32サイクルは、15 MHzの速度で発生します。

## 2タイミング制御

Vircon32システムのすべての動作は、グローバル・タイミング・スキームに従います。タイマーは、60 Hzで「新しいフレーム」信号をトリガーする役割を果たします。その後、各フレーム内で250,000の「新しいサイクル」信号をトリガーします。これにより、1秒あたり15,000,000のサイクルが発生します。

Vircon32コンポーネントによって実行されるすべてのアクションは、常に(直接的または間接的に)タイマーからの信号に対する反応としてトリガーされます。これらの同じタイミング信号はすべてのコンポーネントによって共有されるため、コンポーネント間の調整が必要ないため、全体的なタイミングは大幅に簡素化されます。

### 2.1フレーム内のタイミング

Vircon32システムは、フレームが正確に60 Hz間隔で開始することを保証するために必要です。ただし、フレーム内のサイクルは、リアルタイムで一定の間隔で発生する必要はありません。順序が保持され、250,000サイクルすべてが次のフレームの前に発生する限り、特定の実装は、適切と思われるフレーム内のサイクルを時間設定できます。たとえば、すべてのサイクルを可能な限り高速に実行してから待機することは有効です。その結果、プログラムは、経過フレームサイクルが時間を正確に測定する方法であるとは想定できません。

## 3通信バス

Vircon32コンソール内のチップは、通信のためのメカニズムとしてバスを使用します。これらのバスは、通信を制御し、要求を送信する独自のマスタデバイスがあるマスタ-スレーブモデルに従います。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。両方のコンソールバスにおいて、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。

### 3.1アドレス空間

接続された各スレーブは、一連の32ビットレジスタを公開します。これらのレジスタはすべて、マスターデバイスで使用可能になります。これらのレジスタのそれぞれは、ホストデバイス内で整数値(ローカルアドレス)で識別されます。

複数のスレーブが接続されている場合、バスは各特定のスレーブのローカルアドレスと、マスタが統合グローバルアドレス空間の一部として使用できるグローバルアドレスとの間の変換を担当します。

要求を行うために、バスマスタは、所望のターゲットアドレスをバスに提供することによってターゲットレジスタを選択しなければならない。次に、バスは、要求が転送されなければならない正しいスレーブデバイスを選択し、前記スレーブがその応答を送信するときに同じプロセスを逆方向に実行する。

### 3.2要求メカニズム

どのバスに対しても、CPUがマスタとして実行できる要求には、同じ2種類があります。

* 読み取り要求:

CPUは、特定のグローバル・アドレスに格納されている値の読取りを要求します。要求が成功すると、要求された値が提供されます。

* 書き込み要求:

CPUは、指定された値を特定のグローバルアドレスに書き込むように要求します。要求が成功すると、要求されたアドレスに値が書き込まれます。

どちらの場合も、CPUへの応答には、要求が成功したか失敗したかを示すブール値を含める必要があります。

要求は2つの理由のいずれかで失敗する可能性があります。1つ目は、要求が存在しないアドレス(現在接続されているデバイスのいずれによっても公開されていないアドレス)を参照している可能性があることです。アドレスが存在する場合、スレーブデバイスがその特定の要求を実行できない場合に、2つ目の失敗の原因が発生します。典型的な例は、カートリッジなどの読み取り専用メモリに書き込もうとすることです。

## 4メモリバス

これは、メモリー(RAMまたはROM)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り、メモリー空間はグローバル・アドレス範囲を受け取り、その内容にCPUがアクセスできるようになります。CPUがこのバスのアドレスの読み取りまたは書き込みを要求すると、成功した場合、ターゲット・デバイスのメモリー内の特定のワードの読み取りまたは書き込みが実行されます。

メモリバスの接続は下図のようになっていますが、4つのコンポーネントすべてが常に接続されていても、カートリッジまたはメモリカードがコンソールに接続されていないと、コントローラチップにアクセス可能なメモリがない場合があります。

メモリバスは32ビットアドレスを使用しており、グローバルアドレス空間とローカルアドレス空間の間の変換に使用されるプロセスは、次に示すように、フィールド内のアドレスを分割することです。

接続された4つのデバイスは、次のデバイスIDを使用してメモリアドレスにマッピングされます。

| デバイスID | 接続されているデバイス |
| --- | --- |
| 0 | RAMメモリ |
| 1個 | BIOSプログラムROM |
| 2個 | カートリッジプログラムROM |
| 3個 | メモリカードRAM |

## 5コントロールバス

このバスを使用すると、プロセッサは接続された各チップの一連のI/Oポートにアクセスできます。これらのポートはそれぞれ、チップが公開する単一ワードの制御レジスタです。これらのポートを読み書きすることによって、CPUはさまざまなコンソールチップに操作を要求し、その状態に関する情報を受信できます。

CPUがI/Oポートの読み取りまたは書き込みに成功した場合、一般的な影響は、単に「値の読み取りまたは書き込み」の期待される結果とは大きく異なる場合があります。ポートは、読み取りまたは書き込み時に副作用を引き起こす可能性があります。一部の書き込み要求では、値がまったく書き込まれない場合もあります。特定のポートの動作を知るには、各チップの仕様にある制御ポートのリストをお読みください。

コントロールバスの接続構成は下図のようになりますが、この場合、これらのチップはすべて常時接続されているため、コントロールポートは常にアクセス可能です。

制御バスは、プログラム命令から抽出されるため、14ビットのアドレスのみを使用します(第7章のフォーマットを参照)。グローバルアドレス空間とローカルアドレス空間の間の変換に使用されるプロセスは、次に示すように、フィールド内のアドレスを分割することです:

接続された7台のデバイスは、次の表に示すように、8つの使用可能なデバイスIDにマッピングされます。

| デバイスID | 接続されているデバイス |
| --- | --- |
| 0 | タイマー |
| 1個 | 乱数発生器 |
| 2個 | グラフィックチップ(GPU) |
| 3個 | サウンドチップ(SPU) |
| 4個 | 入力コントローラ |
| 5個 | カートリッジコントローラ |
| 6個 | メモリカードコントローラ |
| 7個 | (デバイスが接続されていません) |

## 6チップコマンド

前述したように、プログラムの実行中、CPUは他のチップから機能を呼び出すために制御バスを利用する必要があります。これを可能にするためにコンソールチップによって使用されるメカニズムは、CPUが特定のコマンドコードを書き込んでそのチップ内の特定のアクションをトリガーできるコマンドポートを公開することです。

#### コマンドパラメータ

一部のチップ機能では、特定のパラメータを指定する必要があります。例えば、プログラムが画面上にテクスチャ領域を描画しようとする場合、どの領域を描画し、画面上のどこに配置するかを知る必要がある。そのコマンドの論理構造は次のようになります。

GPU.DrawRegion(領域,X,Y)

コマンドを記述してチップ関数を呼び出すと、コマンドはパラメータを受け取ることができないため、このタイプの構造は使用できません。代わりに、各チップには、呼び出されたコマンドの動作を制御する一連の内部変数が用意されています。これらの変数は、制御ポートを介して公開されるため、読み取りや変更が可能です。このようにして、領域を描画するプロセスは、代わりに次のような構造に従います:

GPU.Region=SelectedRegion

GPU.DrawingPointX=X

GPU.DrawingPointY=Y

GPU.描画領域()

CPUはまず必要な変数の値を調整し、それからチップのコマンドポートに書き込む。これにより、チップの機能が望ましい方法で動作することが保証される。

## 7内部データ形式

コンソールコンポーネント内で処理されるすべてのデータは、常に32ビットワードの形式です。ただし、同じ32ビットが、コンテキストに応じて異なるコンソールコンポーネントによって異なる方法で解釈されます。このセクションでは、Vircon32システム内で使用可能なビットフォーマットと解釈について説明します。

### 7.1整数

ワードは、32ビット符号付き整数として解釈されます。2の補数表記を使用します。これは、Cのデータ型"int32\_t"と同じです。

これが使用可能な唯一の整数データ型であることに注意してください。符号なし整数は存在せず、サイズの異なるバリアントも存在しません。

### 7.2ブール値

前に説明したように、ワードは最初に整数として解釈されます。次に、この数値はブール値のtrue/false値として解釈されます。整数値が0(つまり、すべてのビットが0)の場合、ワードはfalseとして解釈されます。その他の値はtrueとして解釈されます。これはCのデータ型「bool」と同じです。

いつ生成ブール値。Vircon32は常に、「真」の場合は値1を、「偽」の場合は値0を使用します。

### 7.3フロート

ワードは32ビット浮動小数点数として解釈されます。IEEE 754で定義されている通常の指数表記に従います。これはCのデータ型「float」と同じです。

floatは、NaN(数値ではない)、無限大、不定などの特別な値を使用できることに注意してください。Vircon32システムでは、これらの非数値に対するサポートやエラー検出を実装する必要はありません。したがって、プログラムでこれらの値を使用すると、不確定な動作が発生します。

### 7.4バイナリ

ワードは、互いに独立しており、グループとしてそれ以上の意味を持たない、32ビットのシーケンスであると見なされます。

この解釈は通常、AND/ORなどのバイナリ演算でのみ使用されます。これは、Cがこのような演算で整数を処理する方法と同じです。

### 7.5 GPUカラー

GPUがワードを使用して画面カラーを生成する場合、そのワードはRGBA空間のカラーコンポーネントを表す4つの8ビットフィールドのセットとして解釈されます。これは、多くのグラフィックカードで使用されているRGBA8形式と同じです。

場合によっては、アルファコンポーネントが適用できないことがあります。その場合、アルファフィールドの値は単に無視され、「完全な不透明度」として扱われます。

### 7.6 SPUサンプル

SPUがワードを使用してサウンドサンプルを出力する場合、SPUはそれを16ビットの符号付き整数のペアとして解釈します。1つは左のスピーカー用で、もう1つは右のスピーカー用です。これら2つは一緒に単一の32ビットステレオサンプルを形成します。

この図では、各サンプルの暗い色のビットは、その16ビット整数内の最上位バイトを表します。これは、WAVファイルで使用されるサンプル形式と同じです。

### 7.7 CPU命令

CPUがワードを命令として実行しようとする場合、それは一連の相補的なフィールドを有する命令オペコードとして解釈する。これらのフィールドの中には、制御バスによって使用される「ポート番号」がある。

他のフィールドの使用と解釈に関しては、CPU仕様でカバーされます。

## 8エンディアン

すべてのVircon32システムでは、最小(かつ唯一の)データサイズはバイトではなく、32ビットワードです。これと同じ原理がメモリ内のストレージにも適用されるため、アドレスはワードオフセットとしてインデックス付けされ、個々のバイトにはアクセスできません。

ワードはバイトのシーケンスとは見なされないため、ほとんどの場合、適用する必要のあるエンディアンネスの概念はありません。ただし、ほとんどのコンピュータでは、データはバイト単位で保存およびアクセスされます。このため、バイナリの互換性を確保するために、一部の実装の詳細ではエンディアンネスを考慮する必要があります。

このため、Vircon32システムは**リトルエンディアン**これは、現在のほとんどのコンピュータシステムで使用されている表現です。

## 9ハードウェアエラー

プログラムの実行中に、いくつかのハードウェアコンポーネントは、完了できない操作を実行するように要求される場合があります。一般的な例は、存在しないメモリへのアクセスや、ゼロによる除算などの数学的エラーです。コンソールがこれらの状況を検出すると、ハードウェアエラーがトリガーされます。ハードウェアエラーがトリガーされると、コンソールはプログラムの実行を停止し、エラーを処理するために制御をBIOSエラールーチンに転送します。

BIOSエラー・ハンドラは、エラーに関する基本的な情報を表示し、すべての実行を停止するためにのみ必要であるため、非常に単純です。画面上のエラー表示の例を次のスクリーンショットに示します:

すべてのハードウェア・エラーはCPU命令の処理によってトリガーされるため、エラーのリストとその具体的な処理については、CPU専用の仕様パート3で説明します。

(パート2の終わり)