NES DOOM 的な技術の解説

はじめに

本解説は Norix がファミコン/NES における NES DOOM 的な技術を大雑把に解説するために作成しました。理解の一助になれば幸いです。

本解説の内容に付いて正確性を保証をするものでは無い事を予めご了承下さい。

NES DOOM とは

DOOM は id Software が作成した FPS で、1993 年に PC DOS 用としてリリースされ、1990 年後半からソースコードが GPL ライセンスの元で公開されています。

新しいデバイスをハックして DOOM を動かしてみた猛者現る…というネットニュースを見た事があるかたもいらっしゃる事でしょう。

NES DOOM は 2019 年に Andrew Tait 氏によって作成され、Raspberry Pi3A+と EZ-USB FX2LP を使用して NES で DOOM を滑らかに動かす事に成功しました。現在は GitHub（https://github.com/rasteri/PiPU）にて公開されています。

日本国内においては NES DOOM をファミコンで使えるようにした基板をばくてん（@bakuten_do）氏が作り、家電のケンちゃんでデモが行われネットニュースにもなりました。

ファミコン/NES のバス構成について

まず初めに、ファミコン/NES（以降はファミコンとだけ書きます）のバス構成についての説明をします。

ファミコンはアーケードゲーム基板の規模をバランス良く縮小したような作りになっており、プログラムとキャラクターの ROM をカートリッジに収めており CPU バスと PPU バスの二種類のバスがカートリッジに引き出されています。（他にバンク切り替えなどの機能を持ったマッパーやバックアップ SRAM などがカートリッジ内に入っている場合もあります）

この構成は当時の家庭用ゲーム機としては割と珍しく、一般的な家庭用ゲーム機ではプログラム ROM だけでした。

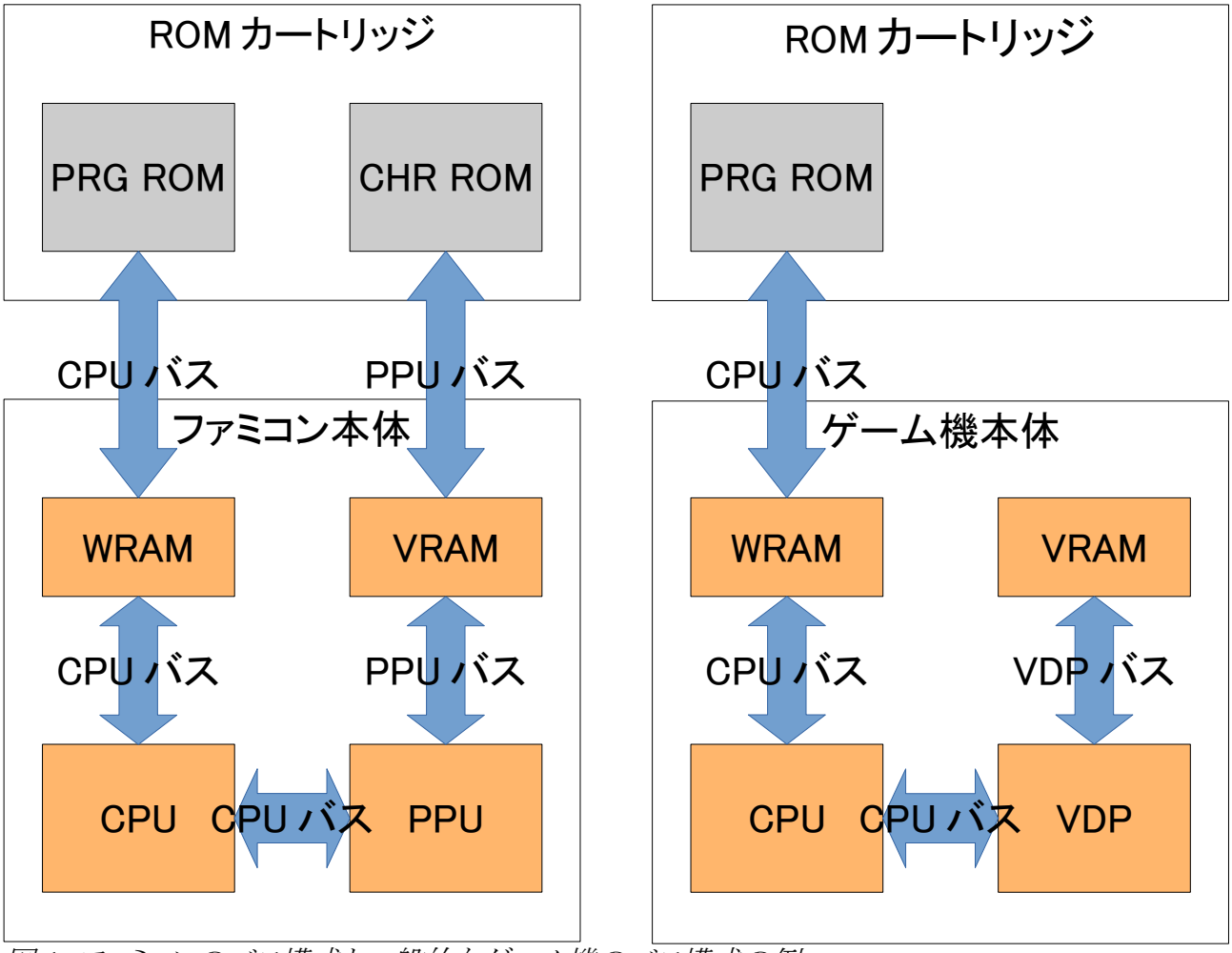

図 1: ファミコンのバス構成と一般的なゲーム機のバス構成の例

ファミコンの PPU バスには PPU が描画する際のバスアクセスの殆どが現れるため、そこを細工してやると多彩な画面効果を得ることが出来ます。

VRAM へのアクセスをカートリッジ側で乗っ取る事まで出来るため、VRAM の内容を ROM に持っておき、バンク切り替えすることで少ない CPU 処理で画面全体を書き換えたかのような効果を出す事も出来たりします。（例：ファミコン版アフターバーナー）

NES DOOM のブロックダイアグラム

下図は NES DOOM のブロックダイアグラムです。

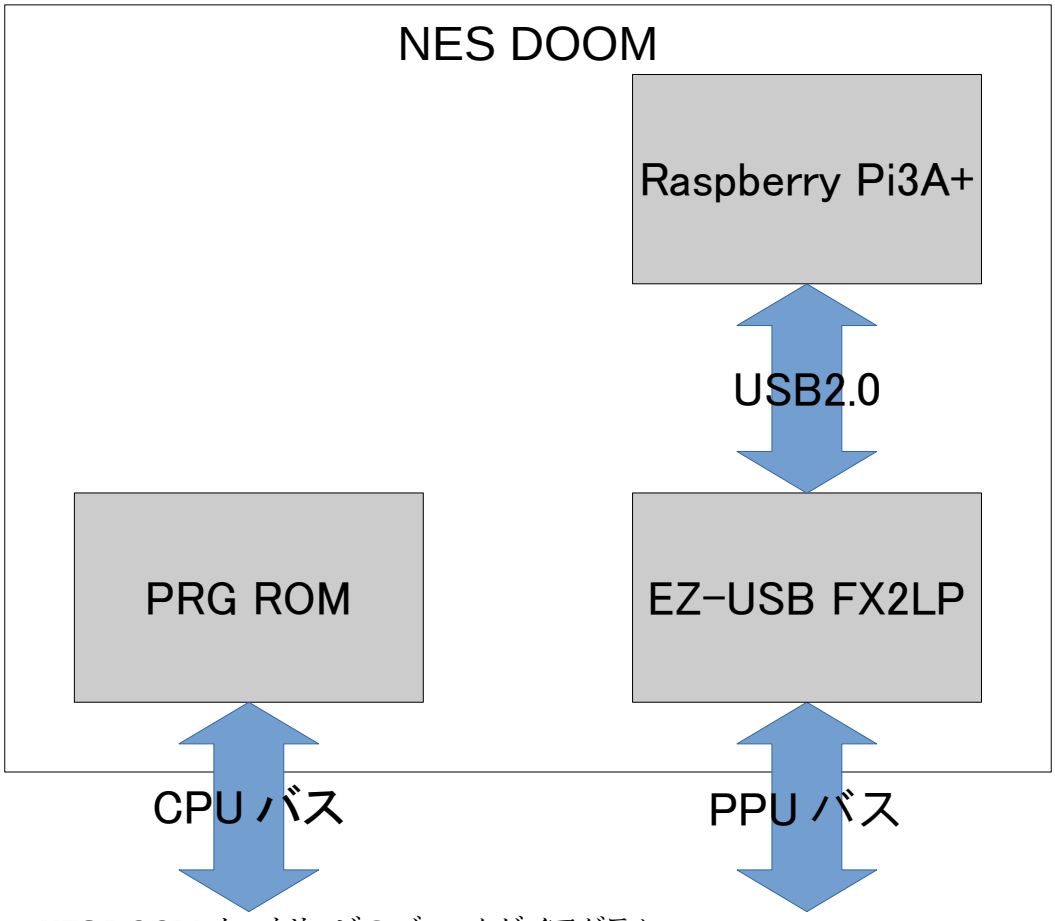

図 2: NES DOOM カートリッジのブロックダイアグラム

以下、ざっくりソースコードを眺めた限りで恐らくこうであろうという感じです。

NES DOOM では PPU のデータ線を乗っ取っており、Raspberry Pi3A+（以下 Raspi）で PPU データバスに流れるデータを生成して EX-USB FX2LP（以下 FX2LP）の内蔵 FIFO（読み/書き用に二つの FIFO があります）を使って PPU とのデータをやり取りしています。なお、PPU バスのアドレス線は未使用です。

FX2LP は 5V トレラント機能があるうえ（Raspi の GPIO は 3.3V で 5V 入力は不可）、80 系のバス形式の/RD と/WR で FIFO の読み書きが可能です。実に NES DOOM に都合が良いチップがですね。

ファミコンの CPU はコントローラの読み取り、PPU のレジスタ及びパレット設定、内蔵音源レジスタへの書き込みが主な仕事で、NMI 割り込み（垂直帰線期間割り込み）に入ると PPU を介して FX2LP からデータを読み取ったり FX2LP へデータを書き込んだりしています。

Raspi は DOOM のプロセスを動かし、別プロセスのスレッドで DOOM プロセスへキー操作を送り込んだり DOOM の画面をファミコンに合うように加工しながら送受信しているようです。

バスアクセスの細かい内容は後述します。

コラム：80 系バスと 68 系バス
バスといっても車両の話ではなく、Intel 80xx（80 系）と Motorola 68xx（68 系）のバス信号形式の話です。
最近の CPU になるまでに多様なバス形式が存在していますが、8bit ～ 16bit 時代の CPU は大別するとこの二種類のバス信号形式に分類出来ると思います。
両者のバス形式の大きな違いとして、読み込みと書き込みのタイミング信号線が分離している（80 系）か 1 つにまとめているか（68 系）の違いです。
面白いことに、ファミコンの場合は CPU が 68 系で PPU が 80 系という組み合わせです。

ネームテーブルとアトリビュートとキャラクタと

PPU のバスアクセスの細かい内容の前に PPU が画面に表示する方式について簡単に説明します。

一般的に古めのゲーム機のバックグラウンド画面（以下 BG）はタイルマップ方式とも呼ばれています。ファミコンの場合は BG にスクロール機能もあります。

ファミコンの BG にはタイルマップに相当するネームテーブルと言われる VRAM があり、いわゆる 8bit パソコンのテキスト VRAM と同様の仕組みです。キャラクタはその名の通りキャラクタ ROM に書かれている文字や記号だと思えばイメージ的にあっているかなと思います。

キャラクタ ROM を RAM にすると PCG になります。ファミコンではキャラクタ RAM のパターンも存在し、代表例としてはディスクシステムがキャラクタ RAM です。8bit パソコンだと X1 の PCG や MSX の SCREEN0～2 が似たようなものでしょうか。

アトリビュートはネームテーブルの属性情報でファミコンの場合はパレットの選択として使われており、アトリビュートも VRAM にあります。ファミコンにはありませんがものによってはアトリビュートでキャラクタの反転などが出来たりします。

詳細を知りたい場合は NESDEV wiki にある PPU の解説（英語）を読むと良いでしょう。

PPU のバスアクセスについて

詳細は NESDEV wiki の PPU Rendering（英語）を読めばわかりますが、ここではざっくりと説明します。

PPU が画面を表示する際、スキャンライン毎に PPU バスは特定のパターンで読み出しを繰り返しています。（NTSC_Timing も参照するとより分かりやすいと思います）

1.ネームテーブル読み込み

2.アトリビュート読み込み

3.タイルパターン読み込み（LOW 側プレーン）

4.タイルパターン読み込み（HIGH 側プレーン）

基本は上記 1～4 の繰り返しで、スキャンライン毎の読み込みは 170 バイトです。1 フレームで 241 ライン分の読み込みをしますのでトータルで 40970 バイトです。

スプライトパターンの読み出しもタイルパターンと全く同じですが、ネームテーブルとアトリビュートの読み込みは使用されません。

ファミコンではスクロール機能の関係でひとつ前のスキャンラインの最後のあたりで左側 2 タイル分先読みしています。

スプライトはスキャンライン毎にそのラインに表示されるかを毎回走査しており、そのラインに表示される場合はプライマリ OAM からセカンダリ OAM へスプライト 1 つ分の 4 バイトをコピーします。セカンダリ OAM はスプライト 8 個分のメモリ領域しかないため、スプライトの 9 個目以上は表示されません。スプライトのタイルパターンを読み込みはセカンダリ OAM を元にタイルパターンを読み込みます。そのスキャンラインでスプライトを走査した後にタイルパターンを読み込むため（タイミングとしては水平帰線期間中になります）、スプライトのタイルパターンが実際に表示されるのは次のスキャンラインになります。

※OAM：Object Attribute Memory の略。スプライトの事をオブジェクトと呼ぶ流派という認識で問題なし

NES DOOM 方式ではこのうち BG の 2～4 の部分だけを使っています。

固定であろうがスクロールさせようが結局は**読み出しするアドレスが違うだけ**なので、スキャンラインやフレーム単位で見ると特定パターンの読み出しの繰り返しです。「読み出しするアドレスが違うだけ」ここ大事なので二回書きました。

つまり、PPU バスのアドレス線を使用しなくて済んでいるのは特定パターンでデータを PPU バスに流してやれば画面が構成出来るからです。ネームテーブル読み込みをしたデータはタイルパターン読み込みをする際のアドレス線を変化させるためのデータなのでアドレス線を使わないから適当なデータでも構わない訳です。

なお、互換機は PPU バスの動作が純正ファミコン PPU とは若干ながら違っていることが多く、MMC5 カートリッジが正常動作しないことが多いのはその若干の違いが MMC5 の正しい動作に必要な部分だったりするためです。

NES DOOM 方式は純正ファミコンの PPU バスの動作に完全に依存しています。よって、MMC5 カートリッジ同様に互換機では動作が怪しいでしょう。

NES DOOM 方式の弱点

NES DOOM ではスプライトは利用されておらず、BG のみで画面を構成しています。NES DOOM はスプライトが無くても構わないのでそれでいいのですが。

また、NES DOOM では PPU バスのみを利用しているため CPU の OAM DMA が利用出来ません。そのため、スプライトの属性情報（位置やアトリビュートなど）を転送するためのクロック数が垂直帰線期間に確保出来ないのです。（工夫すればスプライトも使えなくはないと思います）

単純に計算してみてましょう。

ファミコンの垂直帰線期間は大雑把に 2200 クロック程度あります。BG やスプライトの表示中はそのクロック期間内で PPU へのアクセスを全て終了させるのが基本的なルールです。その期間を外れると、VRAM を正しく読み書き出来なかったり画面が乱れるなど様々な不具合に遭遇してしまいます。

スプライトの属性情報は 1 つのスプライトで 4 バイトあり、64 個のスプライトで合計 256 バイトあります。CPU が PPU からのスプライトの属性データを 1 バイト分転送するのに最低でも 8 クロック掛かります。256 バイト×8 クロックで最低でも 2048 クロック掛かってしまいます。そうすると、垂直帰線期間の残りクロックが 200 クロックもありません。これでは他の処理をする時間が余りにも少ないため、他に何かをしようとすると簡単に垂直帰線期間をオーバーしてしまいます。

CPU の OAM DMA を使えば（ほぼ）514 クロックで済むのとは大きな違いです。

DESTROY THEM ALL の NES DOOM 方式の改良

拙作 DESTROY THEM ALL（以下 DTA）では NES DOOM の弱点であるスプライトも利用出来るようにしたものと考えて差し支えありません。

下図は DTA カートリッジのブロックダイアグラムです。

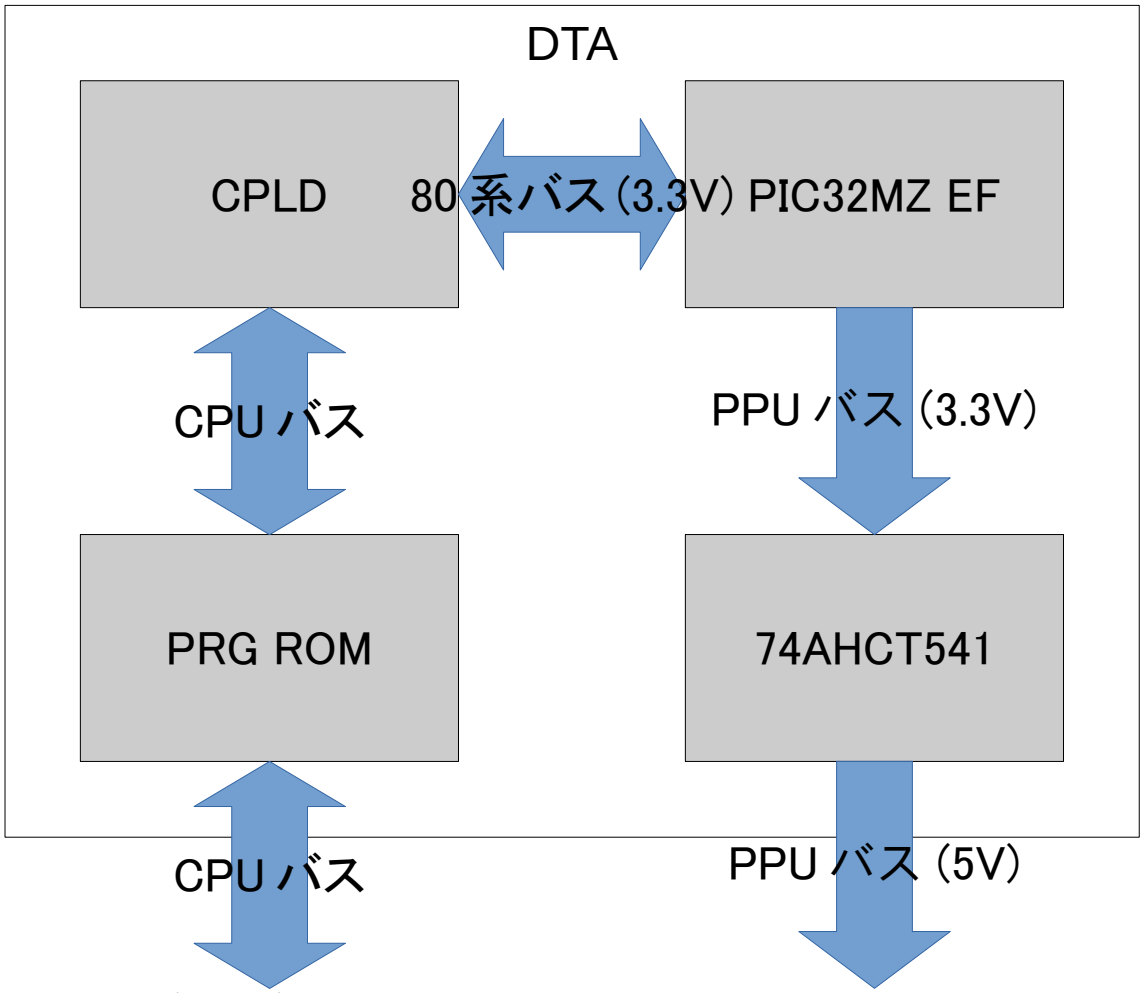

図 3: DTA ブロックダイアグラム

DTA では Raspi と FX2LP の代わりを PIC32MZ EF（以下 PIC32）で実現しています。使用している PIC32 は MIPS アーキテクチャで L1 キャッシュ（インストラクション、データ）があり、最大 252MHz（DTA では 200MHz で使用）で動作する中々にパワフルな MCU です。（エラッタも何だかんだ多いのですが）

PIC32 は 3.3V デバイスで 5V トレラントは一部のピンでしか対応していません。CPLD は 3.3V デバイスですが 5V トレラントがあるものを使っています。CPLD はアドレスデコーダと CPU の 68 系バスを 80 系バスに変換する機能を担当しています。やっていることは単純なので CPLD でなくとも実現可能です。

PIC32 のペリフェラルに PMP（パラレルマスターポート）というものがあり、これを拡張スレーブモードとして利用し、CPU とのインターフェースとして使用しています。この機能のおかげで CPU OAM DMA が利用出来るようになっています。（参考：PIC32 PMP のデータシート）

PPU バスは読み出しのみで利用することにし、PPU /RD を PIC32 のピン変化割り込みで検知し、PIC32 の GPIO に DMA でデータを出力しています。74AHCT541 は単純なバスバッファで、3.3V → 5V への変換と

PPU /RD で GPIO からのデータ出力を制御しています。常にデータが出力したままのものを各所で出力があるバスに繋ぐのは基本的に NG です。

PIC32 の DMA には各種割り込みイベントをトリガーとして CPU を介さず動作開始する機能があり、CPU 負荷が殆どない DMA 転送が使えるようになっています。上手くはまればかなり便利な機能です。

また、DTA は PIC32 のタイマー 2 つと DMA と PWM を利用して 32KHz の PCM を出力しています。今回はファミコンの内蔵音源は使用していません。

ファミコンの CPU はコントローラの読み取り、PPU のレジスタ及びパレット設定、OAM DMA が主な仕事で、NMI 割り込みに入ると PIC32 とデータのやり取りをしています。

DTA の BG レンダリング

NES DOOM 方式では通常のファミコンとは違い 8 ドット×1 ドット単位でパレットを指定することが可能です。イメージ的には MSX の SCREEN2 です。

DTA では PPU へ表示データを常に DMA で転送するため、表示データはダブルバッファを採用しています。

表示するための BG データの保持方法としては以下の方法が考えられます。

1. タイルマップ

2. プレーンビットマップ

3. ピクセルビットマップ

NES DOOM では 3 のピクセルビットマップを変換して転送用データを作成していると思われますが、DTA は基本的にタイルマップ方式を採用しました。タイルチップは 8×8 単位、1 ピクセルにパレット込みで 1 バイトのパックドピクセルとしました。タイルチップがパックドピクセル方式なのは BG を回転する時の計算がしやすいからというのもあります。

ASCII 文字は BG キャラクタ形式でデータを用意しており、表示する際に PPU への転送バッファへコピーするだけで済むようになっています。

タイトル画面もタイルマップ方式ですが、キャラクタはパックドピクセルではなく BG キャラクタ形式と同じもので無駄に変換しなくて済むような形式になっています。

ぶっちゃけ、PPU への転送データ形式に出来ればどれでも可能なので、DTA ではシーンごとにレンダラーを用意して使い分けています。

DTA のスプライトレンダリング

DTA の NES DOOM 方式ではスプライトも使えるようにしたとありましたが、スプライトのレンダリングは少々面倒です。

「PPU のバスアクセスについて」で軽く触れましたが、PPU のスプライトタイルパターン読み込みはスキャンライン毎にスプライトの走査を行った後、コピーされたセカンダリ OAM を元に水平帰線期間あたりでタイルパターンを読み込みます。

つまり、PIC32 側でも PPU のスキャンライン毎にスプライトがスキャンラインで表示されるかのエミュレートを行わなければいけないという事です。

早い話が NES エミュレータのスプライト表示部分とやることに大差はありません。PPU がやってる事なので二度手間なのですが表示されるデータを PPU バスに流す方式なのでやらざるを得ません。

副作用としてですがエミュレートする際にタイルパターンの加工をすると色々な効果を追加する事が出来ます。DTA では地味ですが回転面の表示エリアでスプライトをクリッピングしています。

どうやって DTA を作ったりデバッグしたのかについて

初めて NES DOOM を見たあと、使っているパーツから NES DOOM 方式を大体推測出来たので、NES DOOM 的なハードウェア構成について暇なときに他の方法で実現出来そうな方法をぼんやりと考えていました。

まずは FPGA で SFC mode7 を持った高機能マッパーを作れば出来るであろう事は以前から考えていました。しかしながら、ざっくり回路図を書いてる途中でやたらと規模が大きくなりそうなのでやめました。第零部完。

次に比較的高速な MCU で NES DOOM 的な機能が実現出来ないかを模索しました。PIC32 の PMP を以前に実験したりで知っていたので、それで一応出来そうかな？と思ってデータシートを眺めたり情報を探っていたのですが、どうやら DMA との相性が悪く上手く動かないという質問がチラホラ見つかる。

こりゃダメかもわからんな…と、ぶっちゃけしばらく放置してました。

んで、夏頃だったか何かティン！と来たんですよ。ティンと。

PPU バスへは GPIO に DMA して、CPU バスに PMP 使えば NES DOOM にスプライト出したりまで出来んじゃね？足りないところは CPLD でも使えばいいだろー。みたいな。

思い立ったら吉日なので早速回路図を考えながら作って基板をアートワークして…数日熟成します。いいと思ったので基板屋さんに発注。時々バグを見つけたり改良案が浮かぶので熟成意外と大事にしてます。

必要な部品もポチる。

MI68 までまだまだ時間あったんで…実装してテスト開始するまで一ヶ月くらい放置してました。ネタなので別に出来なくても私は一向に構わん！くらいの勢いで。

グダグダしつつも実装したのでまずは実験です。

自作の Kazzo 的な謎デバイスに簡単なデバッガ機能を付けてあったのでそれでまずは CPLD と MCU が正しく動作するかの実験をしてました。二日ほど試行錯誤の結果、問題なく動作しているようであった。（透明ケースから見える 1 本のワイヤーはこの時のデバッグの結果）

そしてまた数日放置プレイ。

やる気ゲージが溜まったので、実証実験として映像を出すための実験を開始。まずは表示用の画像を二つほど用意して PPU バスに流すデータを作るだけのツールをでっちあげ、謎ツールで動作確認。

で、いざ実機で表示！…してみたものの何かおかしい。画像そのものやでっちあげたツールがバグってたので逆変換するツールもでっちあげてデバッグ。無事に実機で思った通りの画像が出たのであった。第一部完。

ここまでが基礎実験。そしてまた数日放置。やる気ゲージって大事よね。

んで、ゲーム自体をどんなもんにしようかぼへーっと考えてたのですが…

1. アサルトっぽい何か
2. メタルホークっぽい何か
3. スーパーマリオカートっぽい何か
4. GF とか ABII っぽい何か
5. パワードリフトっぽい何か

をぼんやり考えてました。

ゲームを作る際、画像や音楽って用意するの大変ですよね…。音楽はとりあえず置いておくとして、画像は描いては捨て描いては捨てしてとりあえず 1.のアサルトっぽい何かを作ることに決定。他のも考えるだけは考えてたけど。

アサルトっぽい何かを作り始める前に、考えたり捨てたりしながらライブラリをとりあえず作る。

ゲーム本編のデバッグをするのに実機では手間が掛かりすぎるので、まずは PC 上でシミュレート出来るようなものをざっくり作る。

必要なデータコンバータや絵の用意をしながら作る。

一人制作なのでデータ構造だとか急遽変更し放題！やったね俺！

そんなこんなで一ヶ月ほど掛けてゲームっぽい何かが出来た。どっかで見たようなボスキャラがいるのは内緒だし、どこかで聞いたようなタイトルなのも内緒だ。

実機へのポーティング作業を経て、シミュレータで動くものが出来ていたおかげか実機でも割とさっくり動いたのであった。処理がカクカクする訳でもなく普通にゲームが出来ていい感じである。これなら MI68 ネタとして行けそうだ。

音が無かった状態でとりあえずの完成を見たので、いつも MI68 の出展でお世話になっているなすか氏に動画でこっそり見せる。中々良い反応を頂けたので MI68 前に送って展示してもらう事と相成ったのであった。

音が無いとやっぱり寂しいので、効果音だけでも付けるために素材を用意して PIC32 から PCM で出せるようなライブラリを作る。最初、音が上手く鳴らなくてデバッグしてたんだけど、ペリフェラルクロックが止まってただけだったよ…ライブラリそのものにバグが無くて良かった。

効果音だけでも鳴ると今度は BGM が欲しくなる罠。秘密裡に作ってるのと音楽作曲は出来ないマンなので著作権が切れているクラシックから一部小節をパクって BGM とするっ！パートが全部は鳴らせないだとかもあるけど、ただのパクリではアレなのでドラムパターンだけ入れときました。BGM は OPLL のエミュレータで鳴らしてたんだけど気が付いた人、いるかな？

NX labs ロゴも出るだけでは味気なく感じたので、ぐるんぐるん回して拡大縮小させときました。今度こそ完成。第二部完。

あとは皆さんの知るように MI68 でなすか氏のブースで出展して頂きました。毎度毎度、感謝しかないです。

DTA 自体が試作レベルなので、DTA 基板の量産は今の所は考えておりません。すまん。