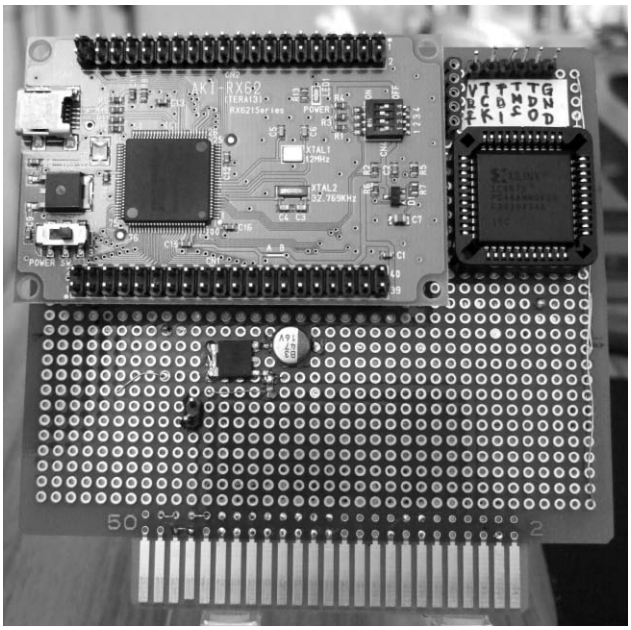
# プロセッサ搭載
# MSX カートリッジの制作

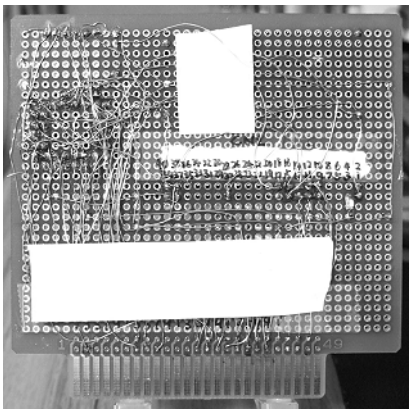


2012 Ryusendo

MSX に 32 ビットマイコンを追加　拡張性の限界を攻めよう

# MSX-RX カートリッジとは？





【写真】カートリッジの表面と裏面。RX ボードは秋月のキット。USB からの電源供給は OFF にし、DIPSW は 1 番から順に OFF-ON-ON-OFF。（1 番は書き込み時 ON）

MSX-RX カートリッジは MSX のカートリッジに ROM ではなくマイコンを搭載した自作の特殊カートリッジです。

今から 10 年ほど前、MSX-H8 カートリッジという自作ハードを考案された方がいました。MSX のカートリッジに H8 マイコンボードを接続し、演算を H8 にやらせるというものです。

H8 はマイコンの中でもローエンドなものですがそれでも MSX の Z80 と比べれば大幅に速く MSX 本体だけでは不可能なパフォーマンスを得ることができます。

もちろん MSX 最大のボトルネックである VDP はまるで変わらないのでアクションゲームをバリバリ作れるとかそういう事にはならないのですが非常に興味をそそられるアイディアでした。

しかし応用例はほとんど発表されていません。理由としては

・考案時、既に MSX は過去のものになっていた
・ハードを自分で作る必要がある
・Z80 と H8 の双方のソフトを開発する必要がある
・肝心の VDP はそのままなのでゲーム作りに向かない

などが挙げられると思います。

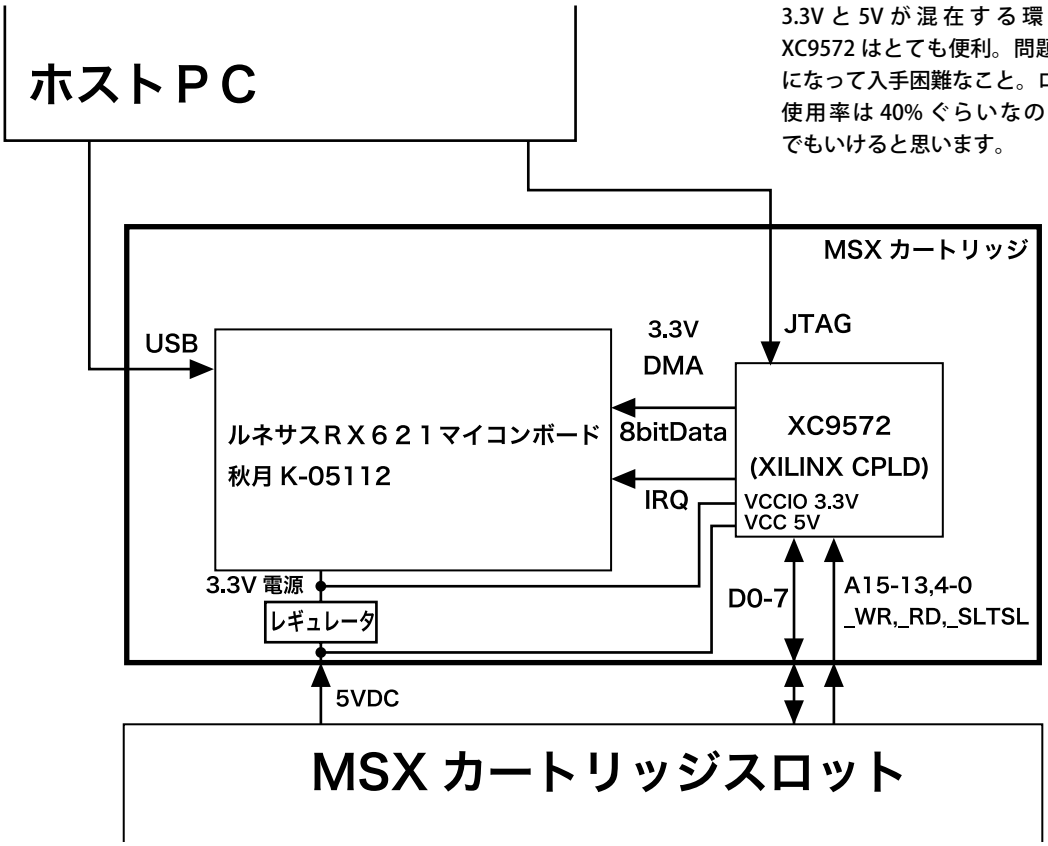
私自身も面白いと思いつつも使い方はなかなか浮かばず、試してみる所まではいきませんでした。

そもそも今 MSX でハードまで付けてどうするんだという思いもありました（^^;)、Chiptune のような「本体だけで工夫する」というジャンルからもはみ出すので邪道っぽい気もしたんですよね。

しかし「いつかは」という思いはずっと持っていました。SFC スターフォックスや MD バーチャレーシングのようなプロセッサ搭載カートリッジは MSX では現れなかった（ゲーム以外ならある）ので「MSX でやったらどうなるか」というのは興味あったんですよね。思いが消えないならやってしまおうと。オリジナルの H8 カートリッジの考案者に敬意を表しつつ、自分なりのものを作ってみることにしました。

回路はシンプル

# 基本構成




3.3V と 5V が混在する環境では XC9572 はとても便利。問題は廃盤になって入手困難なこと。ロジック使用率は 40% ぐらいなので 9536 でもいけると思います。

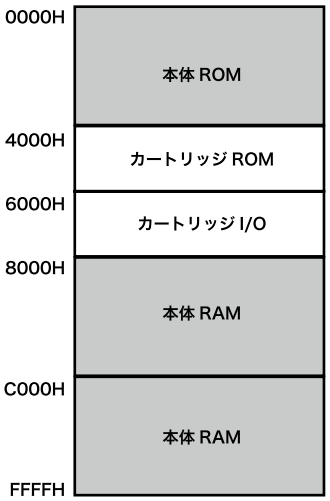
マイコンはルネサスの RX を使います。このマイコンは 96Mhz 動作する上に（マイコンとしては）大容量の 96KB メモリを持つ最新の品種です。

メモリの量は重要です。32KB あれば SCREEN5 の 1 画面分の画像（256x212pixel、4bpp）を記憶できます。マイコン側で画像を完成させて MSX は転送するだけ、という使い方ができます。H8 は外付け RAM で。

RX と MSX の接続には CPLD(XC9572) を使いました。これで細かい回路をまとめられるのでスッキリした基板になります。

双方の通信は RX → MSX には RX の DMA を、RX ← MSX には RX の割り込みで行います。下り方向（RX → MSX）の通信量が大きいので、下りのパフォーマンスに特化した構成になっています。

必要なデータは全てカートリッジ側に持たせるため FDD によるブートは不要です。MSX 側のプログラムは RX マイコンの内蔵フラッシュからロードされます。




カートリッジ I/O
6000H(W)　コマンド送信
6000H(R)　DMA 受信データ
6001～7FFFH(R)
DMA 受信データミラー

MSX 側から見たメモリマップ

基板起こせば標準サイズに収まるはず

# ハード解説

カートリッジはシンプルです。秋月の RX マイコンボードと XILINX の CPLD である XC9572 を使うだけです。

XC9572 は RX と MSX との通信と、MSX の起動に必要な最低限の ROM が入っています。XC9572 はメモリ非搭載ですが、ロジックの組み合わせで ROM と同等なものを用意しています。

ただし非常に小規模な CPLD なので 10 数バイトに収めなければいけません。1 バイトは血の一滴！

XC9572 は 5 VI/O と 3.3VI/O のブリッジという役割もあります。MSX は 5V、RX は 3.3V なのでそのままだと繋げられません。XC9572 は電源 5V、I/O 電圧 3.3V という設定になっています。5V に 3.3VI/O は電圧不足のように見えますが閾値はクリアしてるので問題ありません。また入力（MSX → XC9572）は I/O 電圧に関係無く 5V（電源電圧）まで耐えられる仕様です。つまり特に設定せずとも両電圧共用のピンとして使うことができます。

書き込みは RX は USB、XC9572 は JTAG 経由で行います。JTAG は特殊電子回路社の J-Writer を使いました。

カートリッジには 3.3V の三端子レギュレータが載っています。RX マイコンボードにも同じものが載っていて重複してるんですが、ピンヘッダ経由でボード上のレギュレータに繋げられないのでこうしてます。ボードに直接配線すると見栄えも使い勝手も悪くなるので…

カートリッジのアドレスバスは上位 3 ビットと下位 5 ビットの 8 ビット分しか接続されていません。これはピン数を節約するためです。

**■ ROM の中身**

```
4000 DB 41h,42h
4002 DB 10h,40h

4010 LD HL,6000H
4013 LD (HL),L
4014 LD DE,C000H
4017 LD BC,1000H
401A PUSH DE
401B LDIR
401D RET
```

**6000H に 0 を書き込む（モード指定）ことで DMA から MSX 用のコードが流れてきます。RET で C000H にジャンプします。**

**配線リスト（この他に電源とパスコンが必要です）**

| MSX cart | XC9572 | 秋月 RX ボード |
|---|---|---|
| A15 | 14 | |
| A14 | 13 | |
| A13 | 18 | |
| A4 | 19 | |
| A3 | 20 | |
| A2 | 22 | |
| A1 | 24 | |
| A0 | 25 | |
| _WR | 11 | |
| _RD | 12 | |
| _SLTSL | 7 | |
| D7 | 8 | |
| D6 | 9 | |
| D5 | 6 | |
| D4 | 5 | |
| D3 | 4 | |
| D2 | 3 | |
| D1 | 2 | |
| D0 | 1 | |
| | 44 | INT2(27) |
| | 43 | INT9(18) |
| | 38 | D0(1) |
| | 37 | E7(16) |
| | 36 | E6(15) |
| | 35 | E5(14) |
| | 34 | E4(13) |
| | 33 | E3(12) |
| | 29 | E2(11) |
| | 28 | E1(10) |
| | 27 | E0(9) |

RX ボードのカッコ内は CN1 のピン番号を表します

RX と MSX のやりとり。いちばん重要な部分です。

# 通信方式

先にも紹介した通り下り（RX → MSX）には DMA を使います。具体的には MSX がカートリッジの 6000H を読み出すと RX に DMA 転送要求が送られ、その応答としてポート E（8bit）に転送された値が現れます。ただし転送が行われるのは MSX 側のメモリ読み出しが終わった後なので、MSX には「1 つ前の値」が読み出されます。RX 側で 1 バイト目を手動でセットするか、MSX 側で 1 バイト目を切り捨ててください。以後 6000H を読み出す毎に次の値が読み出されます。

6001H ～ 7FFFH は 6000H のミラーで、どれを読んでも 6000H と同じ結果です。

上り（MSX → RX）は IRQ9 を発生させます。カートリッジの 6000H に書き込むとその値が XC9572 に取り込まれ、同時に RX に IRQ9 の割り込み要求が送られます。RX 側は割り込みルーチンで書き込まれた値を読み込みコマンドとして解釈します。

上りは数バイトのコマンド、下りはキロバイト単位の連続データ、という前提の作りになっています。

速度を上げるために通信手順は簡略化しています。まず DMA 転送が終わったかどうかチェックしていません。RX の DMA はチェックを省くのに十分な速さがあるという前提です。その他の手順も「RX が速いので Z80 が追いつくはずはない」としています。例外が画面描画で、これだけはチェックを行ってます。

FD を不要にする

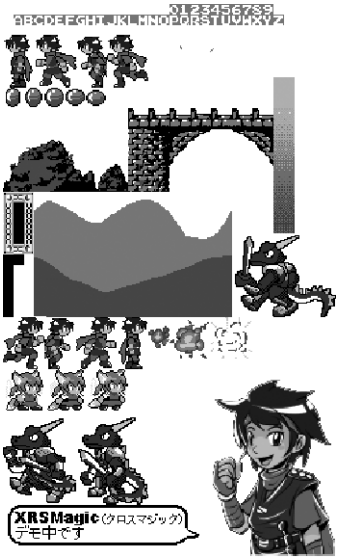
# 起動シーケンス

RX マイコンは MSX の電源投入により起動されます。DMA や割り込みなどの初期化を済ませて割り込み待ちに入ります。MSX のロゴが出る頃には初期化は完了しています。

MSX はまずカートリッジの 4000H と 4001H を読み出し、それぞれ 41H、42H であるかチェックします。その後 4002H，4003H の起動アドレスを読み出しそのアドレスにジャンプします。ROM 部はこの仕様に合わせたものになっています。

ROM 内のプログラムは 10 数バイトの短いもので、RX から MSX の RAM にアプリケーションを転送してジャンプするだけです（前ページ参照）。

以降は RAM 上のプログラムで RX との通信を行います。




**▲デモ用に作ったキャラクター**
**(256x424dot)**

まずは画像処理周り

# RX 側の処理

実際の処理は RX で行います。正直言って今から Z80 アセンブラでアプリ作りはツライので、C コンパイラで作れる RX をメインにしました。

MSX の仕事は出来上がった画像を VRAM に転送するだけです。考え方はシンクライアントに近いですね。

MSX 側ではキー入力や音楽の処理も行います。音楽は既存の MSX 用ドライバを使うので、ここは唯一 MSX が主となる部分です。

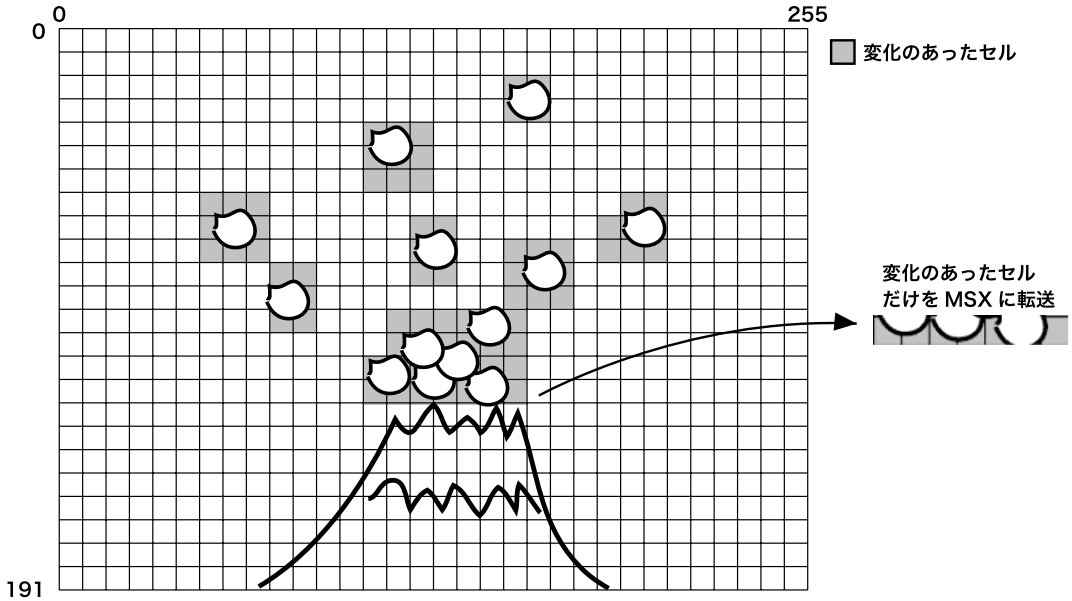
MSX2 の SCREEN5 は 1 画面で 28KB 程度のスクリーンバッファを使います。これを全部転送してもいいのですが、MSX の遅い VDP にはかなりキツい仕事です。出来るだけ転送量を減らしたいところです。

PC-8801 等であれば色数を減らして高速化という手もあるんですが MSX2 では VRAM の構造上できません。描画する領域そのものを減らすしかありません。

そこで画面を８×８ドットのセルに区切って、変化があったセルだけを転送するようにします。

変化を検出するには新旧二つのスクリーンバッファが必要…ですが 96KB 中の 50KB 以上を占拠するのは後々を考えると不安なのでセル毎に 32bit のハッシュを算出してそれを比較する方式にしました。ハッシュが違えばそのセルの変化があったという事です。

欠点としては情報量を 1/8 に減らしてるので誤検出（描き変わってるセルを変化してないと判断）が生まれることですが、算出方法を工夫して目立たないようにしています。




画面全体をスキャンして変化のあったセルだけを MSX に転送する。1 セルは 8x8 ドット。
MSX2 の 1 画面は 256x212 ドットあるがここでは縦 192 ドットに制限して負荷を下げている。

ハード周りを確立させた後のプログラミング

# RX 側の処理２

■矩形描画

2D ゲームの画像処理は、あらかじめ用意したキャラをスクリーンバッファに書き込む矩形描画がメインになります。MSXBASIC で言えば COPY 命令ですね。

またキーカラーを使用した透明ピクセルの処理も重要です。

これを素直に実装すると 1 ドットずつカラーを調べて透明だったらスキップ、不透明だったら 1 ドット描画…ですがさすがの RX マイコンでもこの処理は重荷です。しかも 16 色画面では 4bit ずつの処理なので複雑な処理になります。

この辺は長年の悩みだったんですが最近論理演算だけで処理できることが分かりました。処理速度が飛躍的に上がり、余裕ができました。

■ゲーム作り

ここまでの仕組みを作れば後はゲームロジックを組み立てるだけです。

ゲームロジックは RX で全て処理します。と言っても実機デバッグは手間がかかるので PC 上でシミュレータを作って制作しました。

シミュレータは自作 MSX エミュレータのソースに RX カートリッジの処理を継ぎ足ししたものです（MacOSX 用　非公開）。

ある程度出来上がったらソースを RX の開発環境にコピーして動作チェックします。

RX の書き込みはボード付属の USB のみで出来るので初期投資は少ないのですが、書き込みの度に電源を切ってモード切替スイッチを操作するのが手間です。FPGA のように起動中でも自由に割り込んで書き込みできたらなあと思います。

■まとめ

かなりややこしい事をしてますが一度土台を構築すれば 32bit 環境で MSX の絵を動かせます。int は 32bit だし実数演算も気兼ねなく使えます。64KB の壁もありませんし IDE は HEW（ルネサス純正）だったり GCC だったりで近代的。

レトロ愛好家でも 8bit でのプログラミングにはあまり戻りたくないのが本音じゃないでしょうか。このシステムでは現代の開発環境から 8bit 実機を動かすというユニークな試みを行っています。他の機種への応用も可能です。

# TOPICS

MSX で画面を転送する場合、一番速いのは VDP コマンドの HMMC（CPU → VDP 転送）のようです。VRAM アクセスのほうが単純な分速そうに思えるのですが実際には逆でした。「TurboR でも VDP の速度は変わらないからゲームには無意味」と思いがちですが、TurboR だと HMMC の速度限界まで攻められるので差が出るようです。

開発が一段落したら「とりあえず動くことが確かなバイナリ」をバックアップしておくといいです。ソフトが思い通りに動作しなくなって行き詰まった時に役立ちます。このバイナリで動作がおかしければ…　どこかが接触不良になっているのでしょう。

このペーパー発行直前まで HEW（無償版）の制限を 256KB だと思ってたんですがよく見たら 128KB になってました。サンプルには支障ないんですが、これ以上作るなら SD カードが必要かな…　もしくは GCC で作るかな…

ピン互換はありませんが特殊電子回路社の「RaXino ♪」でも動作するんじゃないかと思います。

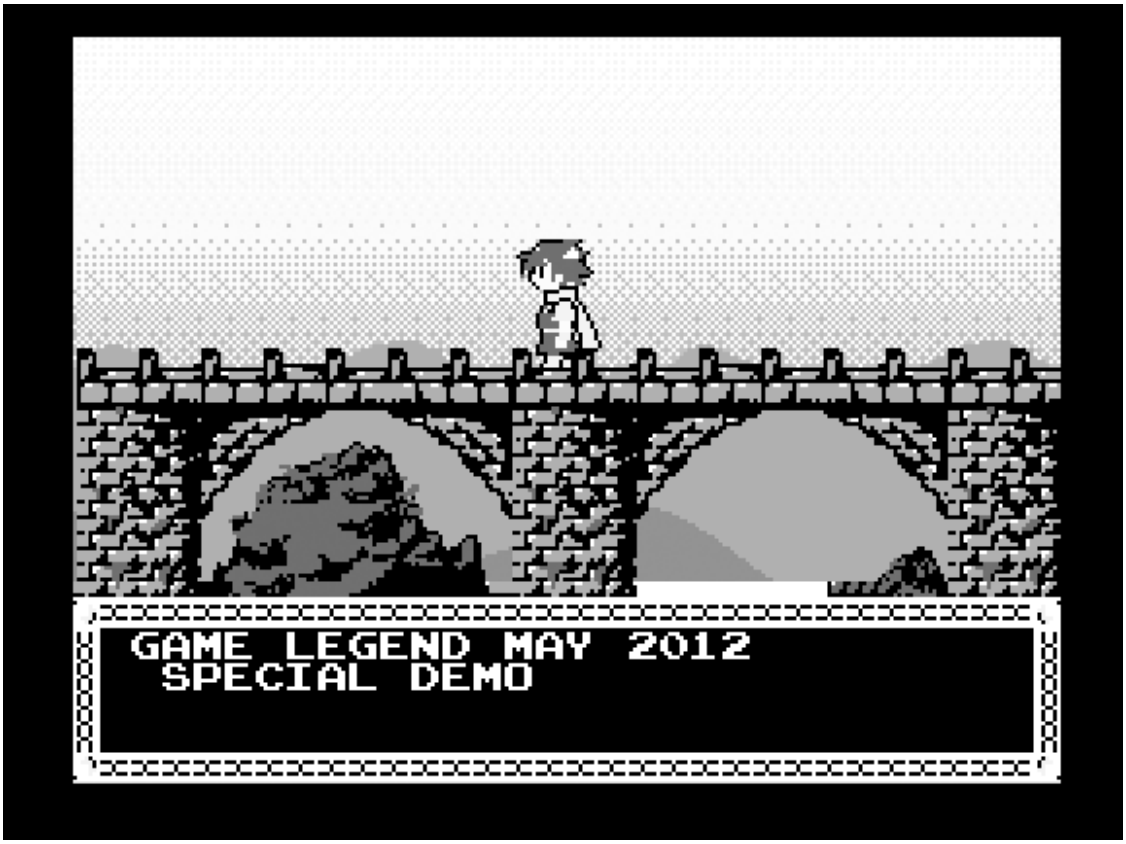
RX マイコンはリトルエンディアンで動作させてます。

画像やパレットなどのバイナリはシミュレータ上に C ソース変換機能を付けています。ソース形式なので開発環境を選びません。

今のところ RX にリセットを付けていないので MSX をリセットしても中の処理は継続します (^^;

静止画だとただの MSX ゲームの画面になってしまうのですが…

# 開発画面




今回一番時間がかかったのはデモプログラムの用意かも…久しぶりのドット絵は大変でした。32 ドットあれば大丈夫と思ってたけど今やるとずいぶん小さく感じます。今の絵は身長 40 ドットです。

STG は作る方が多いのであえてアクションゲーム風にしました。手前の木と橋、奥の背景の 3 重スクロールです。MSX 単体でやらせると凄く重くなるところでも安定した動きを見せています。RX の処理が速いので RX 側の負荷の変化は体感できないという感じです。

スクロールしない場面ではまるでスプライトで動かしてるかのような滑らかな動きになります。

以降は PC 上のシミュレータでソースを作成し、ある程度出来上がったところで実機テストという流れで開発します。実機の無い場所でもノート PC で開発できるので効率的です。

2012 年 5 月 20 日 ( 日 ) 発行
発行サークル：流線堂
著者：Ｒｏｏｔ（nemoto@tri6.net）
ゲームの博物誌：http://ryusendo.rdy.jp/（←現在工事中です） twitter：rsd_rot

本制作記事は進行中のものを扱ってます。仕様は変わる場合があります。
記事内容は無保証です。電子工作の知識がある方向けの内容が含まれています。
制作物の公開予定は未定です。