pio-doc を読む

レトロCPU駆動のためのハードウエアパートを PIO で実現する。そのための読み・メモ書き

実現したい機能

- メモリアクセス開始信号をトリガとして、アクセス遅延ビットを立ててCPUを待たせる。
  - 例: Z80 の MREQ -> WAIT の流れ
- 待たせている間に、有効なアクセスかどうかを判定する
  - 例: Z80 の非 RFSH 検出
  - アドレス上位のデコード
- アドレスデータの読み込み、Readデータの書き出し、Writeデータの読み込みと処理、これはCPUソフトウェアにより行う。PIOは関与しない。
- データバスの準備が整ってから、CPU待たせの解除
- CPUが読み込み完了後、データバスをINmodeに引き戻す。
- 最後に、次のメモリアクセスを待つ。

アイディア

Z80を題材にまず動くものを作る。その過程でPIOに慣れたい。

EMUZ80 エミュレーションがいいだろう。メモリアクセスのみ、アドレス解釈はソフトで実行。 PIOはCPU止めとCPU再開後のデータバス方向制御の2機能を行う。単純かつ本質。

- トリガ: MREQ に wait 命令を掛けて待つ
- RFSH信号がアサートされているときは先に進まない。WAITもかけない
- RFSH信号チェック後、WAITピンをアサートする(Lにする)
- IRQを立てる。ソフト側に通知
- 同時に TX FIFO データ待ち(ソフト処理が完了するとTX FIFOに書き込む)
- TX FIFO にデータが現れると、後半戦スタート
- WAITピンをネゲートする。CPU待たせの解除
- MREQピンが1になるのを待つ
- 1になれば、GPIO0-7を入力モードにする(OUT PINDIR を使用)

ここまでならステートマシン1個でできるんじゃないかな。

オールソフトで実行すると、バス方向切り替えとCPUのデータラッチがレース条件となり、読みそこないが生じる。MREQネゲートを検出してPINDIRを切り替えるやり方は安全である。BUSRQ/BUSACKで同期を取るやり方はみっともない。スマートな処理を外付けハードなしで実現できそうで期待が高まる。

読み込み項目

1. まずはここから

- WAIT 命令でMREQピン指定する
- OUT命令でWAITピンを0にする

2. IRQ立ててソフト実行を再開

- IRQ立てて、
- TX FIFOデータ待ちに入る。

3. TX FIFOデータ待ち

- OUT PINDIR ではOSRから8ビットシフトアウトする。ソフト側でTX FIFOに0xff or 0x0 を書き込んでデータバスの方向をソフトに指定させることにする。というか、この場所では常にPINDIRはin方向だ。