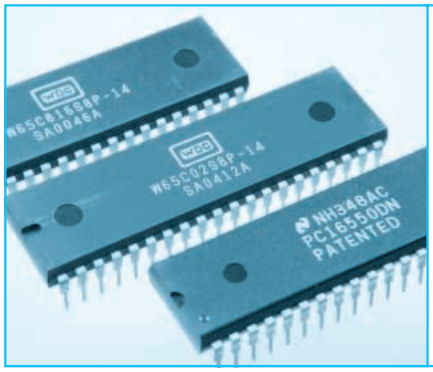
連　載



マイコン・システムのしくみを基礎から理解する

6502マイコン・ボード製作記

〈第4回〉メモリ，I/O空間の割り付けとブート・ローダの書き込み

桑野　雅彦
Masahiko Kuwano

連載第3回(2006年7月号)までを整理すると，以下のようになります．

- リセット後は$FFFC, $FFFD番地に書かれたアドレスからプログラムが実行される
- IRQBによるハードウェア割り込みやBRK命令によるソフトウェア割り込み発生時は，$FFFE, $FFFF番地に書かれたアドレスに飛ぶ
- NMI発生時は$FFFA, $FFFB番地に書かれたアドレスに飛ぶ
- $0000～$00FF(ゼロ・ページ)，$0100～$01FF(スタック)領域はRAMにする

以上から，メモリ空間の末尾部分は通常はROM，先頭部分はRAMにすればよさそうです．あとはI/Oデバイス16550をどこに配置するかです．

アドレス・デコードとイメージ・アドレス

● アドレス・デコードとは

CPUが外部バスをアクセスするときは，アドレス・バス(アドレス・ピン)にアクセスしたいアドレスをセットします．外部回路側では，アドレス・バスの状態からどのデバイスに対するアクセスなのかを判定して，該当するデバイスのチップ・セレクト(被選択信号)ピンをアサート(有効に)します．このように，アドレス・バスの状態からデバイス選択信号を生成することをアドレス・デコード，その回路をアドレス・デコーダと言います(図4-1)．

8ビットCPUのアドレス・バスは16本もあるので，すべてをデコードするのは案外面倒です．例えば，I/Oデバイスのアドレス・ピンが4本あり，16バイトのアドレス空間を使用するとします．もし，これを$5600～$560Fの領域にきちんと割り付けようとすると，アドレスが0101 0110 0000 XXXX(Xは不定)となっているときに，16550のチップ・セレクト信号がアサートされるようにする必要があります．

12本の入力状態が0101 0110 0000となっているという判定は，74シリーズで作るとゲートを多段に接続しなくてはならないのでなかなか面倒です．

● イメージ・アドレスとは

このような場合，必要十分なビットだけをデコードするという考えかたから，アドレスの途中のビットの

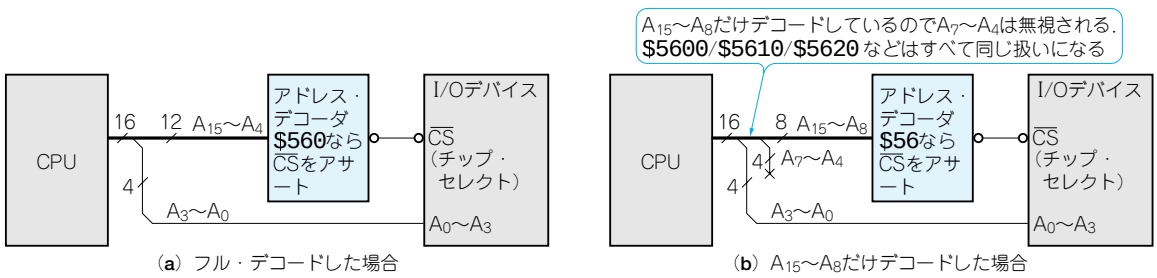


図4-1　アドレス・バスの状態からデバイス選択信号を生成するアドレス・デコード
すべてのビットをデコードせず必要十分なビットだけをデコードすると回路を簡略化できる

Keywords
6502，アドレス・デコード，チップ・セレクト，アサート，アドレス・デコーダ，フル・デコード，イメージ・アドレス，ブートストラップ・ローダ，ブート・ローダ，IPL，Initial Program Loader，フラッシュROM，SRAM，Am29F010，JEDEC，HM628128

デコードを省略するという手法がよく使われます．

図4－1(b)の例で，上位の8ビットぶんの$56だけをデコードしてみます．すると，$5600をアクセスしても$5670をアクセスしてもチップ側から見ればセレクト信号がアサートされ，下位4ビットは0000なので，まったく同じアクセス動作になります．

このように，アドレス・デコードの省略により，異なるアドレスでありながら同じ場所へのアクセスになるものをイメージと呼び，本来配置しようとしたアドレス(予定アドレス)と異なるアドレスをイメージ・アドレスと呼んでいます(**図4－2**)．先の例では，$5670は$5600番地のイメージ・アドレスとなります．アドレスをすべてデコードして，このようなイメージが出ないようにすることをフル・デコードと呼んでいます．

この例ではデコードは上位側を行いましたが，下位側だけ行うという場合もあります．例えば，x86CPUでは，メモリ空間のほかに64Kバイト(アドレス16本ぶん)ぶんのI/O空間をもっています．PCでは，16ビットあるI/Oアドレスのうち下位の10ビットぶんしかデコードしていません．したがって，I/O領域は$0000～$03FFの1Kバイトぶんで，$0400，$0800などは$0000のイメージ・アドレスになっています．

● 上位2ビットだけをデコード

今回は，なるべく回路を単純にするという方針から，アドレス・デコーダもコンパクトにまとめるために思い切って簡略化することにしました．

今回搭載するのはROMとRAM，そしてI/Oとしての16550の三つです．メモリ空間の先頭側($0000側)はゼロ・ページやスタックがあるのでRAMを，末尾側($FFFF側)はリセット・ベクタなどがあるのでROMを配置し，中間部分に16550を配置すればよさそうです．

アドレスの上位2ビットをデコードすると，64Kバイトのメモリ空間を4分割することになり，各エリアは16Kバイトずつになります．

これでとりあえずメモリ・マップが決まりました(**表4－1**)．動作中は基本的にこのメモリ・マップで動かせばよいでしょう．

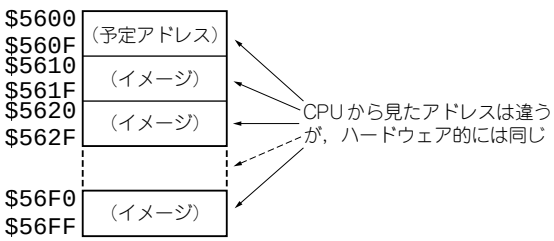


図4－2　異なるアドレスでありながら同じ場所へのアクセスになるものがイメージ
予定アドレスと異なるアドレスをイメージ・アドレスと言う

表4－1　6502マイコン・ボードのメモリ・マップ(途中)

アドレス	用　途
$0000～$3FFF	RAM(16Kバイト)
$4000～$7FFF	空き
$8000～$BFFF	I/O(16550)
$C000～$FFFF	ROM(16Kバイト)

ブート・ローダを手動で入力する

● ROMライタもユニバーサル・プログラマもない!

メモリ・マップは決まりましたが，ROMにどのようにしてプログラムを書き込むのか，という問題があります．

ROMライタやユニバーサル・プログラマがあれば書き込んだROMを実装するだけですが，今回はROMライタも使わない方針です．

● フラッシュROMに通信用プログラムを書き込む

そこでまず，フラッシュROMはどのような手順で書き込むのかについて考えてみます．単純に，アドレスとデータを与えてライト信号をストローブする程度であれば，スイッチを使って手作業で書き込むこともできないことではありません．

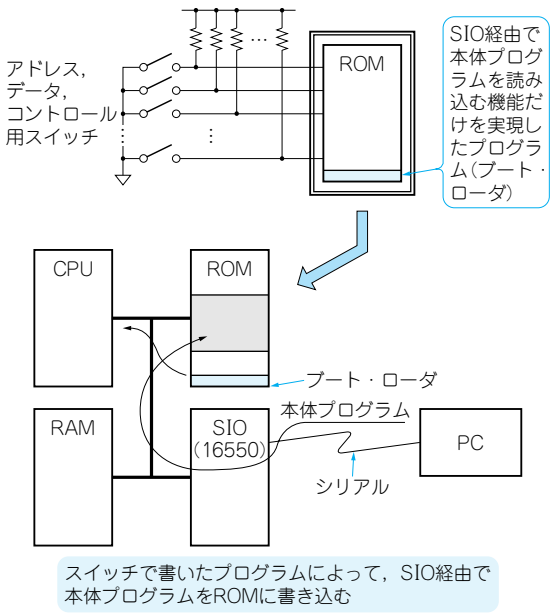


図4－3　フラッシュROMに手作業で本体プログラムを書き込む方法(SIO：Serial I/O)
フラッシュROMに手作業でブート・ローダを書いてから，フラッシュROMに本体プログラムを書き込む