Linuxによる8bitCP/Mマシンの開発

近 千秋 †　　清水 尚彦 ††
東海大学 大学院 工学研究科 †　　東海大学 電子情報学部 コミュニケーション工学科 ††
259−1292 神奈川県平塚市北金目 1117
E-mail:{1aepm024, nshimizu}@keyaki.cc.u-tokai.ac.jp

概要　i8080A 命令互換プロセッサは、組み込みシステムコントローラとして幅広く用いられている。我々は、組み込みシステム開発環境を Linux 上で構築し、この環境の上で i8080A 命令互換プロセッサを開発し、このプロセッサを用いた CP/M マシンを CPLD ボード上で実現した。Linux 上の開発環境の構築方法と、組み込みシステムの開発手法について述べる。
キーワード：HDL, i8080A, CP/M, 組み込みシステム

A Development of an 8bit CP/M Machine by Linux

Chiaki Kon †　　Naohiko Shimizu ††
Graduated School of Engineering, Tokai Univ., †　　School of Engineering, Tokai Univ., ††
E-mail:{1aepm024, nshimizu}@keyaki.cc.u-tokai.ac.jp

Abstract　It has been popular that i8080A compatible proseccor uses as an embedded system controller. We made an embedded system development environment on Linux system. Then we made i8080A compatible proseccor in this environment. And we made a real workable CP/M system with this CPLD based processor. In this paper, we explain how to make an embedded system development environment on Linux system and an embedded system.
Keyword: HDL, i8080A, CP/M, Embedded System

1 はじめに

コンピュータシステム開発を行う際、EDA ベンダーの提供する統合開発環境を用いる例が多いが、このような開発環境は大きな経済的コストと性能の高い計算機を必要とし、小規模な研究・開発には大きな負担となる。我々は、ターゲットが小規模な組み込みシステムであれば Linux マシン上でも開発が可能と考え、今回その事例としてハードウェア記述言語を用いて i8080A 命令互換プロセッサを開発し、その実装例として ROM/RAM ディスクベースの CP/M マシンを製作した。

i8080A 命令互換プロセッサは、8bitCPU の代名詞的存在であり、現在でも組み込み分野を中心に幅広く利用されている。現在入手できる i8080A 互換プロセッサの多くは、周辺回路が同じチップ内に集積化され、機能拡張が施された 1 チップマイコンが大半である。当然のこととして、組み込み用途のためにハードウェアの拡張を行うのは容易ではなく、小規模な変更でも複数の LSI を使用しなければならない。そこで、CPU 自体をハードウェア記述言語を用いて開発し、CPLD で実現することで、ハードウェア拡張にかかる手間を大幅に省くことが可能となる。

OS である CP/M は開発元のディジタルリサーチ社の買収によって、権利が Caldera 社に売却されソースコードが公開となり、現在では WEB 上でソースファイルとバイナリファイルを容易に入手することができる [8]。CP/M と i8080A 互換プロセッサで動作させるコンピュータシステムは、ポピュラーな物であり、開発環境が充実している。そこで、開発した CPU 上で CP/M を動作させることで、このコンピュータシステムを再現した。

このシステムの設計からシステムシミュレーションまでを、Linux 上で行なった。今回、既存のいくつかのアプリケーションを連携させることで、パワーオン前に最終的なボード上でのシステム動作までを Linux 上のシミュレーションで確認することができた。この CP/M マシンを例にして、Linux での組み込みシステムの開発手法について述べる。

2 システム構成

製作したシステムの概要を図 1 に示す。CP/M マシンは、ROM/RAM ディスクベースのシステムとして構成する。CP/M マシンを実装するボードとして PALTEK 社の P3−FLEX10K50 ボードを使用する [6]。ボード上の CPLD には RS232C 接続端子が接続されているので、コンソール入出力用として使用する。CPLD ボード上にはメモリ用の IC ソケットがないため、ディスクとして使用する RAM と ROM は別途ボードを製作し、ケーブルで CPLD ボードと接続する。ROM、RAM 共に容量は 1Mbit で、アクセススピードは各 150[ns]、120[ns] である。

ROM はシステムディスクとして使用する。最初の 128Byte にブートローダを置き、その後ろに CP/M イメージを格納する。アドレス 2000H 以降はディスク 1 として使用する。ディスク 1 にはアプリケーションを格納する。

RAM は最初の 64KByte を CP/M のシステムメモリとして使用し、残りをディスク 2 として使用する。メモリマップは図 3 のようになる。

i8080A 命令互換プロセッサ my80 は、ノンパイプライン構成の CISC プロセッサである。ROM/RAM ディスクの扱いを容易にする為にアドレス拡張を行い、拡張アドレスアクセス用の命令拡張を行っている。また、外部ロジックを使用しなくてもよいように、アドレスデコーダ，UART，ブート時の ROM/RAM 切り替え回路を内蔵した。メモリ参照は、1 クロックサイクル中に 1 回行う。このため、動作周波数はメモリのアクセススピードに大きく依存する。このシステムでは、2.304[MHz] のクロックで駆動している。

このシステム構成で CP/M マシンを実現するために行なった CPU の機能拡張について述べる。

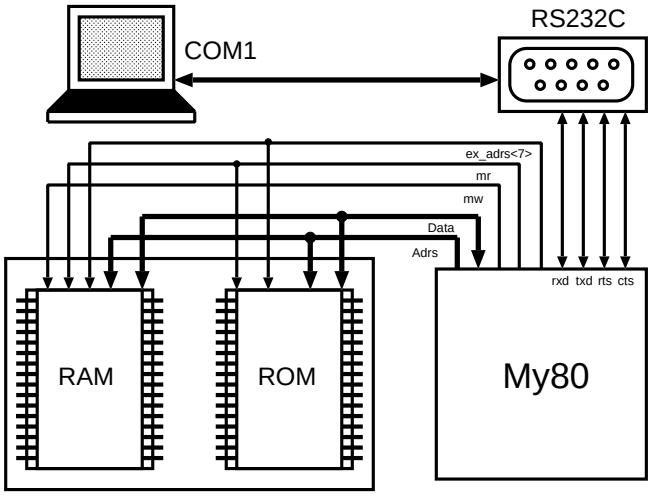

図 1: システム構成

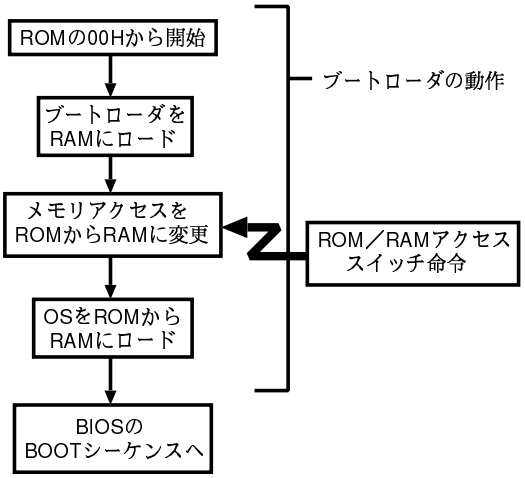

図 2: ブートの流れ

2.1 ROM/RAM アクセスのスイッチ

このシステムのブートの流れを図 2 に示す。電源投入後、まずブートローダの ROM から RAM への書き込み命令を実行し、命令の取り出し先を ROM から RAM へ移行して OS のロードを始める。命令の取り出し先を ROM から RAM にスイッチするため、専用のレジスタと命令の拡張を行なった。ブートローダでこの命令を実行することで、アクセス先をスイッチできる。

2.2 RS232C インターフェース回路

コンソール入出力を RS232C 端子で行うために、インターフェース回路 UART を開発した。通信は 9600[bps] の 8bit ノンパリティで行っている。CPLD は上記の通り 2.304[MHz] で駆動するため内部でクロックの分周を行い、調歩同期で外部とタイミングを合わせる。

入出力を行う際には、入出力バッファのステータスが READY の場合に書きこみ、読み出しを行うため CPU 側からインターフェース回路の内部状態が見えなければならない。そこで、RS232C インターフェース回路のステータスレジスタと入出力バッファへアクセスできるように拡張を行なった。入出力の制御は、この命令を用いて BIOS 側で行う。

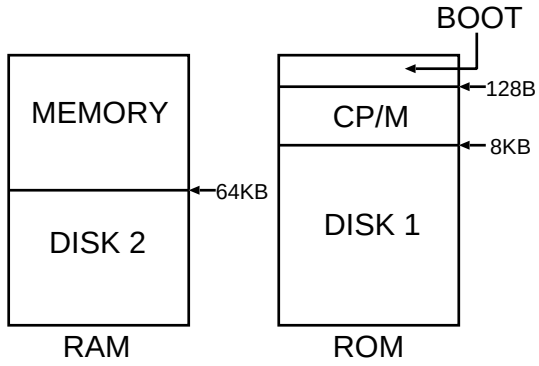

図 3: メモリマップ

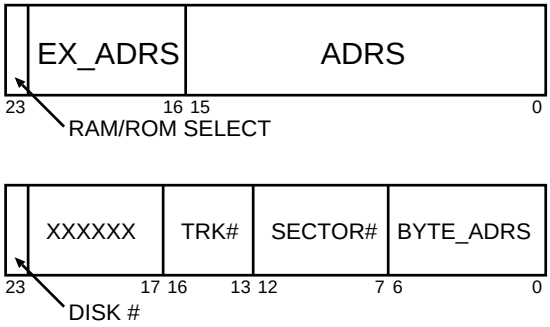

図 4: アドレス拡張

2.3 ROM/RAMディスクへのアクセス

i8080A は 16bit の実アドレス空間しか持たないため、ROM と RAM のディスク領域にアクセスするためにアドレスの拡張を行う。拡張は 8bit とし、最初の 1bit は ROM と RAM の選択 bit とする。拡張アドレスを用いてディスクアクセスするために、BIOSにアドレス計算ルーチンを追加し、拡張アドレスを使用して書き込みと読み出しを行うための命令を追加した。図 4 に示すようにアドレスの拡張を行い、通常の命令動作では拡張部分の MSB を ROM と RAM の選択に使用する。拡張したディスクアクセス命令の場合、B レジスタに拡張アドレス、HL レジスタに通常のアドレスをセットしアドレスバスに出力する。アドレス計算は図 4 に示すような形で行われる。

3 Linux上での開発環境

Linux 上で次のアプリケーションを連携させることで、このシステムの開発を行なった。

- PARTHENON
- yaze
- perl
- pcb

PARTHENON システムは NTT が開発した LSI 設計・開発環境である [7]。PARTHENON 上では、ハードウェアの設計とシステムのデバッグを行なった。BIOS とブートローダの設計・開発及び、ハードウェアデバッグ用の命令コードの生成には Z80 エミュレータ yaze を用いた [9]。この 2 つのアプリケーション間の連携を円滑に行うため、perl 言語でスクリプトを作成しファイル形式の変換を行なった。また、メモリを搭載するボードを製作する必要があるため、基板デザインに pcb を用いてメモリ基板の製作を行なった [10]。各アプリケーションで行う作業について説明する。

3.1 PARTHENON上での作業

PARTHENON システム上では、ハードウェアの RTL 記述、論理シミュレーション，デバッグ、そして BIOS，ブートローダ，ソフトウェアを組み込んだシステムシミュレーションを行う。

まず、各ハードウェアを単一のモジュールとして設計し、各々について論理レベルでの動作シミュレーションを行い、設計の正しさを検証する。今回の場合、プロセッサコアとなる i8080A 命令互換 CPU と、RS232C インターフェースを個別に設計して、それぞれについてシミュレーションを行なった。個別の動作確認後、単一のハードウェアシステムとしてまとめ、拡張する機能を組み込む。拡張機能に関して動作確認を行なった後、実際に BIOS とブートローダを組みこみ、動作の検証を行

う。最後に、ROMに書き込むシステムディスクイメージを組み込み、ブレッドボード上の構成を考慮したコンピュータシステムのシミュレーションイメージを構築する。ここでシミュレーションイメージ上でコンソール入出力を行い、最終的なコンピュータシステムとしての動作を確認する。

3.2 yaze上での作業

yaze上では、BIOSとブートローダの開発及びデバッグを行う。yaze上でCP/Mを動作させることで、開発環境として用いることが出来る。　プログラミングはアセンブラで行い、CP/MのASMコマンドを用いることでHEXファイルとPRNファイルを出力させる。拡張命令以外の命令はi8080A互換でありZ80でも実行可能であるため、yaze上でも等価に実行することが可能である。そこで、CP/MでCPUデバッグ用のシミュレーションプログラムを作成し、DDTコマンドでこのプログラムを実行した際のyazeの動作を確認した後、CPUのシミュレーションイメージに組み込む。DDTコマンドは、CP/Mのデバッグコマンドであり必要であれば各レジスタ値の変更も可能であるので、状況に応じたシミュレーションが可能である。

BIOSとブートローダは、Z80未定義命令が含まれている個所以外はyazeでも実行可能である。そこで、拡張命令部分にはNOP命令を挿入してCP/Mでアセンブルとリンクを行う。yaze上でも実行可能な部分はDDTで動作確認を行なう。ROMのバイナリイメージを作成する場合、DDTコマンドでNOP命令を拡張命令に書き換え動作シミュレーションはPARTHENON上で行う。

CP/Mのファイルイメージは`MOVCPM`コマンドで得ることができるので、今回の場合64KByteCP/Mのイメージを作成し、BIOSとブートローダを組み込む。　システムディスクイメージは、yazeのディスク管理プログラムccdmを用いることで作成す。適切なパラメータを与え、CP/M上で使用したいコマンドをディスクにコピーする。このシステムディスクイメージにCP/Mイメージを組み込み、最終的にROMに格納するディスクイメージが完成する。

3.3 perlスクリプトを用いた連携

CP/M を用いて作成したプログラムを、PARTHENON 上のハードウェアシミュレーションイメージで動作させるため、PARTHENONシステムのシミュレーションプログラム、SECONDS用のスクリプトに変換する必要がある。SECONDSのメモリーへのセットコマンドは、数値での入力を必要とするため、HEXファイルか、CP/Mで`DUMP`コマンドを実行した結果のlogファイルをunixファイルとして保存し、このファイルをSECONDSのコマンドに変換する。ここで、シミュレーションイメージのメモリ配置に対応したアドレス変換と、シミュレーションスクリプトへの変換を自動的行うことで、作業効率が大幅に向上した。

3.4 pcbを用いた基板設計

前述の通り、P3−FLEX10K50ボードにはディスクリートのメモリを配置する場所がないため、メモリ配置用の基板を製作する。基板設計にはpcbを用いた。配線にはスズメッキ線を使う方法もあったが、配線量が比較的多くなるので信頼性の観点から専用基板を製作した。実際に用いる基板の大きさを考慮し、設計を行なった後、感光基板で実際の基板を作成した。

3.5 統合開発環境

以上のアプリケーションを用いることで、Linux上に組み込みシステムの統合開発環境を構築できる。PARTHENONシステムとyaze上のCP/Mを、perlスクリプトによって結合することで、1つのシステム開発環境として使

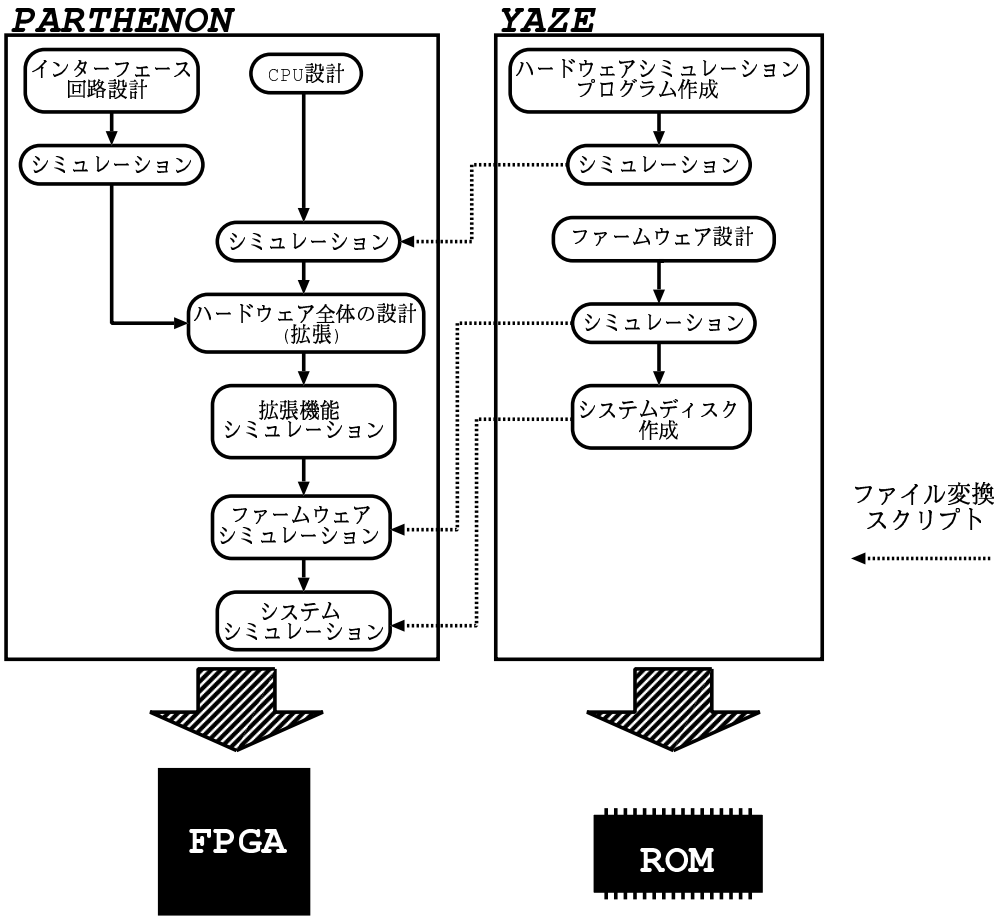

図 5: Linux 上での作業の流れ

用することが可能である。この環境を用いた開発の流れを図5に示す。PARTHENONではハードウェアの開発と、システムシミュレーションを行い、yaze 上の CP/M では BIOS とブートローダの開発を行なう。CP/M が出力するファイルを perl スクリプトで変換し、PARTHENON 上で実行できる形にすることでシステム全体のシミュレーション、評価を行なうことが出来る。この環境を用いることで、ハードウェアとソフトウェアの変更を即座に反映することが可能である。PARTHENONとCP/Mが最終的に出力するファイルは、CPLDベンダツールを用いて、実際のシステムに搭載可能である。

4 SFL記述

4.1 CPU

CPU はノンパイプラインの、CISC プロセッサとして記述してある。1 クロックサイクルに 1 メモリサイクルを実行し、命令実行には 2〜6 クロック要する。PC 更新用の 16bit インクリメンタの設置と、1 クロック /1 メモリサイクルによってオリジナルの i8080A や Z80 に比べて大幅に必要クロック数は少ない。CPU の状態遷移図を図 7 に示す。CPU は最初の if ステートで命令の最初の 1 バイトを `op1` レジスタに格納し、`f1` ステートで命令のデコードを行なって実行の制御を行なう。2 バイト、3 バイト命令の場合は命令フェッチを続ける。各命令の動作を、メモリアクセスを行なうごとに分割して記述し、この動作を各状態

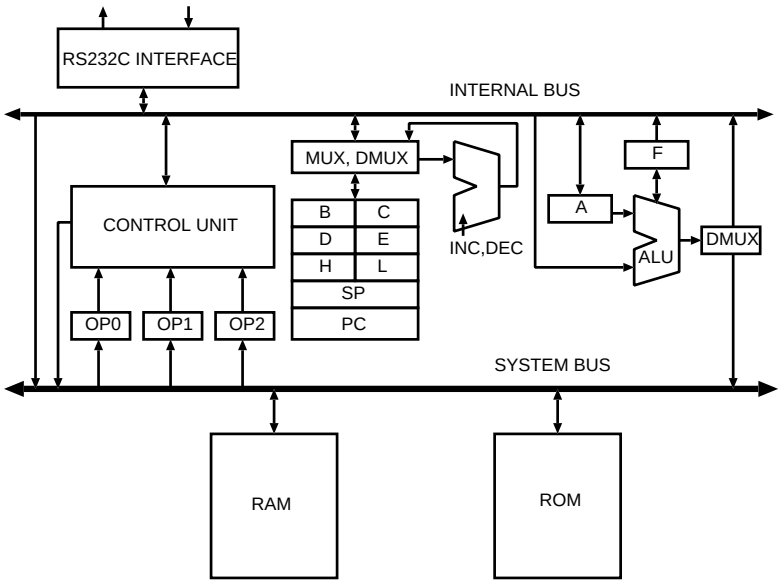

図 6: my80

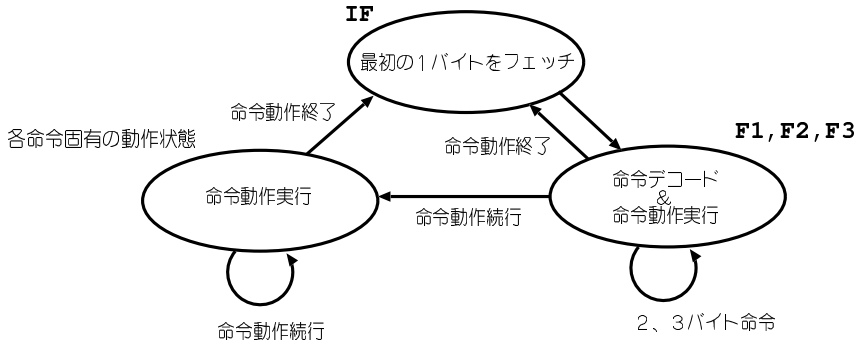

図 7: my80 の状態遷移

に割り付ける。命令を全て取り出した後、1 クロックサイクルで命令が完了する命令は if ステートに戻り、実行に複数クロック要する命令は、固有の状態に遷移していく。メモリからの命令は、OP0 - 2 レジスタでラッチする。OP0 - 2 は命令中で一時レジスタとしても使用している。

目標とする動作周波数が低いため、演算器には高速化の工夫は行なわず、ハードウェアのサイズを押さえるために CPA を用いている。また、インクリメンタとディクリメンタは同じ演算器を兼用している。CP/M の BIOS では割り込みを使用しないため、CPU の割り込み機能は実装しているが使用していない。

4.2 RS232C

PARTHENON で設計したハードウェアは単相同期回路になるので、RS232C インターフェースのために SFL 記述内でクロックの分周を行なう必要がある。クロック周波数が 2.304[MHz] で伝送速度は 9600[bps] であるから、240 クロックに一回送受信できればよい。しかし、この方法で送受信を行なうと、各 bit の最初の部分を値として取ってくることにな

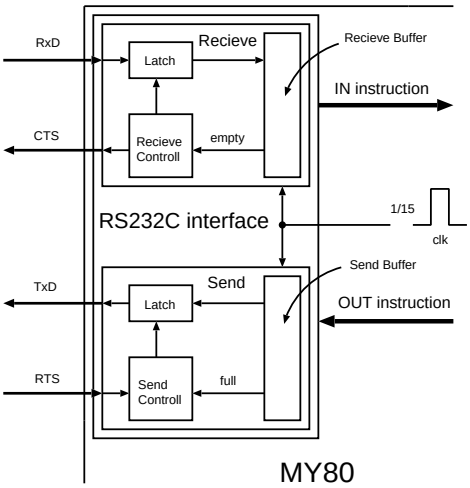

図 8: RS232C インターフェース

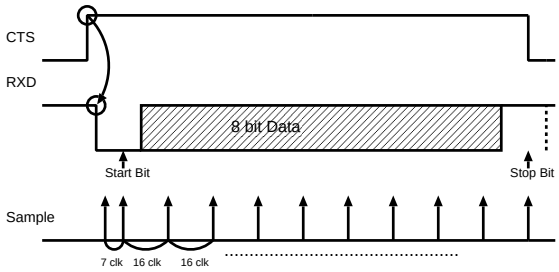

図 9: インターフェースのタイミング

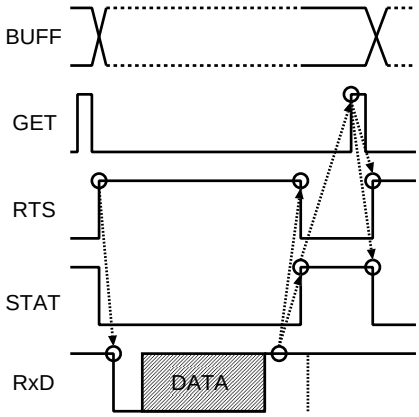

図 10: 受信のタイミングチャート

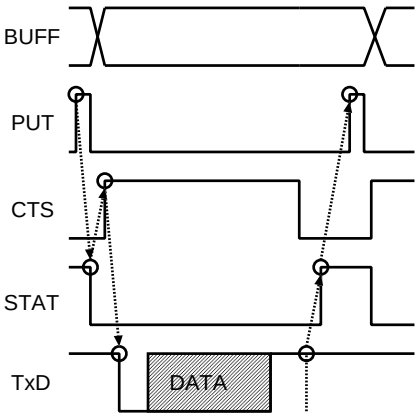

図 11: 送信のタイミングチャート

り、誤りにつながる可能性がある。そこで、このタイミングを更に 16 分割する。図 9 の様にスタートビットを検知してから最初に 7 クロック待った後、16 クロックに 1 回のタイミングで値を取ることで各 bit の真中を取ってくることができるようにする。このタイミングで動作させるには、UART はマスタークロックの 15 分の 1 のクロックで動作させればよい。

通信は、データ 8bit、ストップビット 1bit のノンパリティ通信で行なう。シリアルデータは、LSB から MSB の順に送受信する。ハードウェアハンドシェイクには、RTS、CTS、RxD、TxD 線を用いる。送受信データバッファは 1 エントリとした。

データ受信のタイミングを図 10 に示す。受信側の RTS が Hi であるとき、送信側が RxD にスタートビットを出力することで通信が始まる。データの送信が終了し、RxD にストップビットが出力されたことを確認した後、受信バッファのステータスを有効にし、同時に RTS の出力を Lo に落とす。受信バッファのデータが読み込まれると、受信バッファのステータスを無効にし、RTS を再び Hi にする。

データ送信のタイミングを図11に示す。バッファのステータスが有効であるとき送信バッファにデータが送られてくると、バッファのステータスを無効にし送信待ち状態に入る。CTS の入力が Hi であれば、TxD にスタートビットを出力する。その後、送信バッファのデータを LSB から MSB の順番で TxD に出力する。全

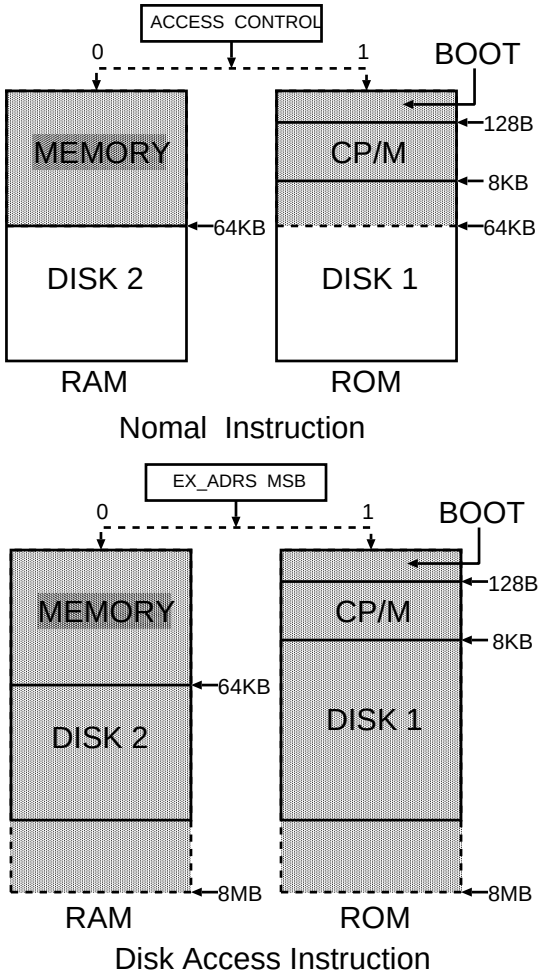

図 12: アクセス制御とメモリ空間

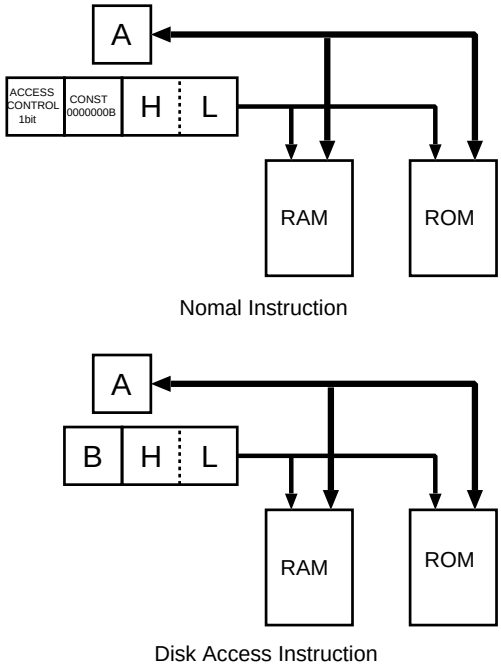

図 13: ディスクアクセス命令動作

てのデータを出力した後ストップビットとして、TxD に Hi を出力し、送信バッファのステータスを有効に戻す。

以上のタイミングと仕様を満足するように設計を行ない、モジュール同士で通信を行い設計の正しさを検証していった。

4.3 ハードウェア拡張

システム構成の検討を元にして、必要なハードウェアの拡張を行なった。各拡張部分について、その拡張方法を説明する。

4.3.1 ROM/RAM アクセス制御

ROM と RAM のアクセス制御のために 1bit のレジスタを設け、この値を拡張アドレス部の MSB に出すことで、通常のメモリアクセスでは自然にアクセス先の変換を行なえる。ブートローダにおいて、ブートローダ自身を RAM にコピーした後、このレジスタ値を変えることでメモリアクセス命令の参照先が ROM から RAM に切り替わる。

4.3.2 拡張レジスタへのアクセス

入出力命令において、ポート番号の MSB が 1 の場合にレジスタへのアクセスとすることで

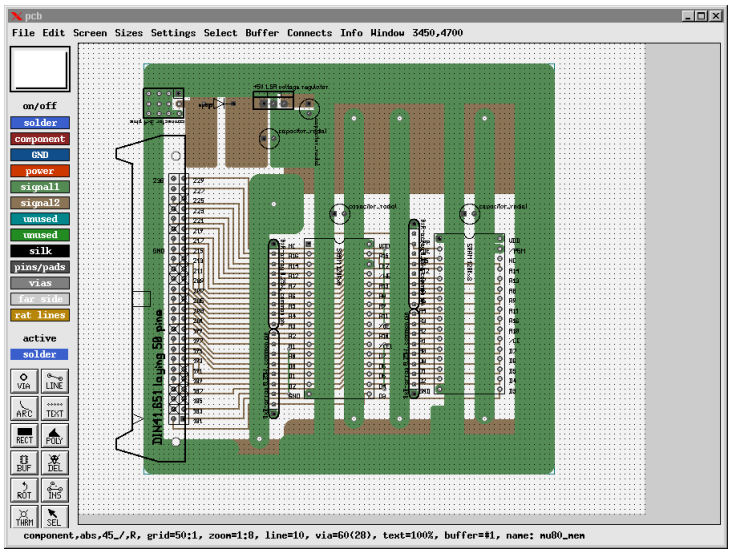

図 14: メモリ基板

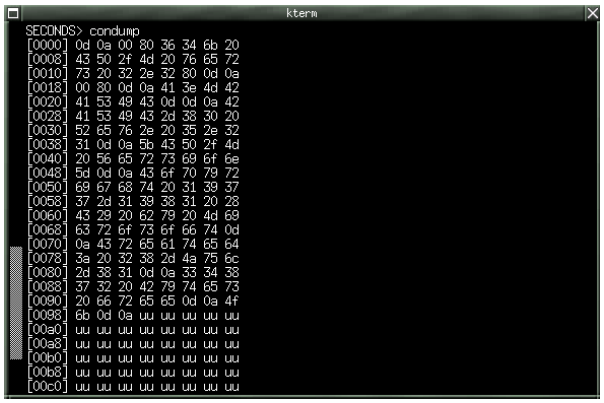

図 15: シュミレーションの出力

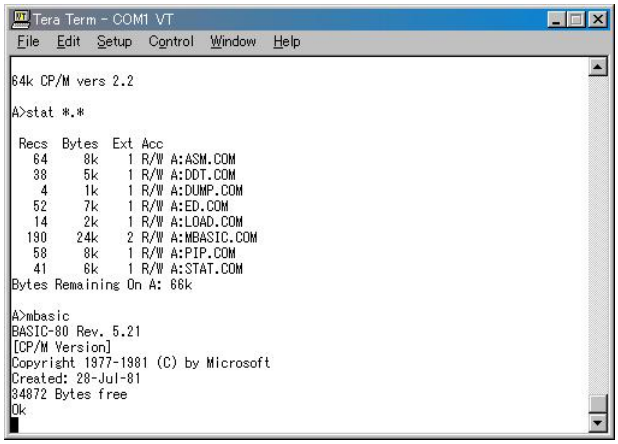

図 16: 実システムの出力

対処した。これでハードウェア内に設けた送受信バッファ、送受信ステータスレジスタ、アクセス制御レジスタといった、拡張レジスタにアクセスする。

4.3.3 ディスクアクセス

ROM/RAMディスクへのアクセスのためには、拡張アドレスバスへアドレスを出力しなくてはならないため、ディスクアクセス命令を CPU に追加する。追加命令は 2 バイト命令とし、最初の 1 バイトでディスクアクセス命令が判明し、2 バイト目でロードかストアかが判明する。ここで 2 バイト目をデコードする必要があるので、記述のデコーダ部分を拡張しておく。アドレスは、事前にディスクアドレス計算ルーチンによって B、H、L レジスタにセットしておくので、命令動作としては図 13 のように A↔[BHL] というかたちになる。アドレスバスの拡張部分へは、図 12 に示すように通常のメモリアクセス命令では ROM/RAM アクセス制御レジスタの値を MSB に出力し、残りの bit へは 0 を出力する。ディスクアクセス命令の場合のみ、拡張アドレス部に B レジス

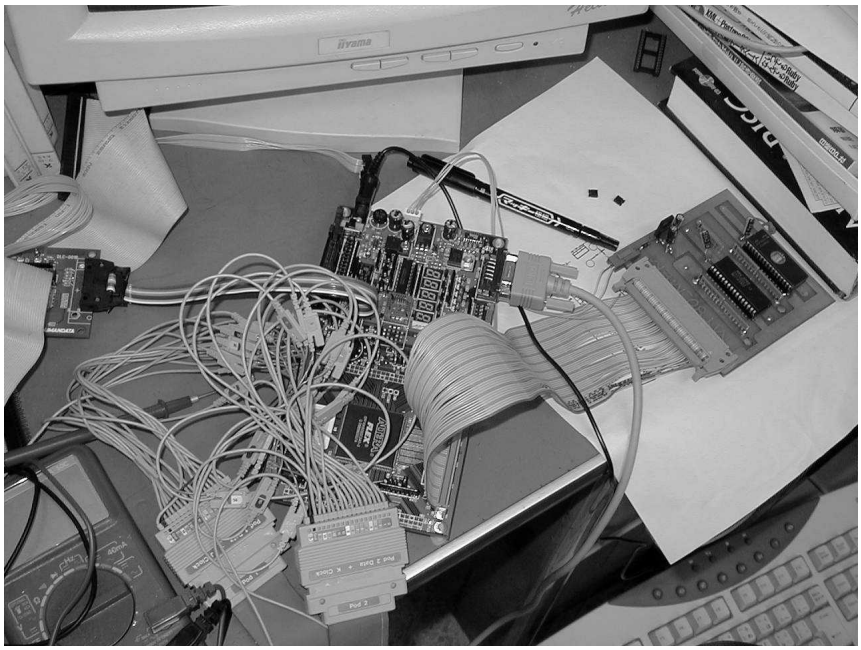

図 17: 実際のシステム

タの値を出力する。通常のメモリアクセス命令では、16bit のメモリ空間しか見えないが、ディスクアクセス命令の場合ディスク 1、2 それぞれ 8MB までアクセス可能である。これらを図 11 と図 12 に示す。

5 CP/M マシンのシミュレーションと評価

前節で示した環境を Linux 上に構築し、CP/M マシンの製作を行なった。

まず、CP/M でデバッグ用プログラムを作成し、これを用いて i8080A 互換であるハードウェアのプロトタイプの開発を行なった。my80 が i8080A 命令互換プロセッサとして動作することを確認した後、RS232C インターフェースと命令の拡張を行ない、ハードウェアシステムのシミュレーションイメージを構築した。このシミュレーションイメージを用い SECONDS 上で CP/M を動作させ、実際のコンソール出力を確認するシミュレーションを行なう。実際のハードウェア構成を考え、コンソール側には、my80 の送受信相手となる RS232C インターフェースと、コンソール出力を確認するために 00H からアドレス順に、コンソール出力が入っていくメモリを配置した。システムディスクの内容を SECONDS のメモリーセットの形式にインポートし、SECONDS 上でブートローダからの OS のブートを行ない、組み込みコマンドをコンソール上で実行するシミュレーションを行なった。コンソール側のメモリに格納された値は図 15 のようになり、この値を ASCII コード表と対応させ、コンソール画面に出力されている文字列を推定した。結果として、システムが正常に動作し、組み込みコマンドも実行できていることを確認することができた。

システムシミュレーションで動作が確認できたところで、実システムへのハードウェア、ソフトウェアの搭載を行なった。my80 の評価を ALTERA 社の MAX+plus2 上で行ない、最大動作可能周波数が約 11[MHz]、使用ロジックセル率が、FLEX10K50 デバイスで 43[%]、FLEX10K30 デバイスで 74[%] という結果を得ることができた。実際の動作クロックは 2.304 [MHz] であり、デバイスは FLEX10K50 であ

るので、実装するためには十分な結果が得られた。P3−FLEX10K50ボードと作成したメモリ用基板を用い、実システムを組み上げ、PC上のターミナルソフトで操作を行った。結果として、図14に示されるように、実際にこのコンピュータシステムが動作していることが確認できた。

6 まとめ

Linux上に組み込みシステムのハードウェアとソフトウェアの統合開発環境を構築し、この環境上でCP/Mマシンの開発を、CPUの開発から、システムシミュレーションまで行なった。このCP/Mマシンを実システムとして製作し、実システムでの動作も確認することができた。このように、複数の既存のアプリケーションを用い、アプリケーション間の連携をスムースに行なうためのプログラムを追加することで、Linux上にCPLDハードウェアを含めた組み込みシステムの開発環境を構築することができる。開発環境を低コストで実現できるため、学校機関や個人ユーザなどの小規模なプロジェクトにおいては非常に有効な手段であると考えている。

現在の状況では、最終的なCPLDへのマッピング段階で、CPLDベンダの用意するツールを使わなければならない。ここで、オープンなCPLDのアーキテクチャが存在すれば、デバイスへの書き込みまでオープンソースのアプリケーションで行なうことができる。今後の課題として、このようなCPLDデバイスまでを含めた検討を行なっていきたい。

参考文献

[1] http://shimizu-lab.et.u-tokai.ac.jp/

[2] intel "MCS-86 User's Manual"

[3] 安居他"CP/M手作りマイコン"産報出版

[4] 村瀬"実習CP/M"ASCII

[5] http://www.atmarkit.co.jp

[6] http://www.paltek.co.jp

[7] http://www.kecl.ntt.co.jp/parthenon

[8] http://www.retroarchive.org

[9] http://www.moria.de/~michael/yaze-p2dos/

[10] http://bach.ece.jhu.edu/~haceaton/pcb/

[11] 清水"Linux/Alpha 活用講座" Linux JAPAN 2001年8月号 五橋研究所