卒業論文

リアルタイム制御用マイコンの回路と開発環境の設計

指導教官　鈴木正國教授

北川章夫助教授

秋田純一助手

電気情報工学科　４年

集積回路工学研究室

数馬晋吾

平成１１年２月２４日

目次

第１章　研究の背景と目的 1

第２章　設計仕様 2

２－１　命令 3

２－１－１　命令体系 3

２－１－２　命令の動作の詳細 4

２－２　実現方法 6

２－２－１　入出力の説明 6

２－２－２　各モジュールの説明 7

２－２－３　命令動作時のフロー 15

第３章　開発環境について 21

第４章　チップへの実装 23

４－１　回路記述とシミュレーション 23

４－１－１　VerilogHDL による回路記述 23

４－１－２　Verilog-XL によるシミュレーション 23

４－２　論理合成とシミュレーション 24

４－２－１　Design-Analyzer による論理合成 24

４－２－２　Verilog-XL によるシミュレーション 24

４－３　配置配線とデザインルールチェック 25

４－３－１　Apollo による配置配線 25

第５章　まとめ 27

謝辞 28

参考文献 29

第１章　研究の背景と目的

回路の集積度が上がり、一つのチップ上に搭載出来る回路の規模は日に日に大きさを増してきている。それに従い、今まで別々のチップ上で実現されてきた回路を集積し、一つのチップ上に載せることが盛んに行われている。特に、民間の企業では既に CPU にタイマや PWM（Pulse Width Modulation）、EEPROM（Electrically Erasable Programmable ROM）までを搭載したチップの例があり、製品化もされている。さらに、そういったチップの場合、通常企業がアセンブラやＣ言語などの開発環境までを整えるのが一般的になってきている。

しかし、大学でのチップの試作においては、DSP(Digital Signal Processor)や特定の機能だけをテストするための回路など、特殊用途のものがほとんどで、汎用的に使えるものが試作されることは少ない。

そこで本研究では、CPU や PWM、タイマ、そしてプログラム用のメモリ(EEPROM)までを一つのチップ上に載せること、並びにそのチップ上に走らせるプログラムの開発環境であるアセンブリの作成を通じて、ハードウェアとソフトウェアを同時に開発する際にどういった事が問題となるのかを検証する。

なお、本チップ試作は東京大学大規模集積システム設計教育研究センターを通し ローム(株)及び凸版印刷(株)の協力で行われる予定のものである。

第２章　設計仕様

今回設計するチップの特徴を以下に挙げる。

・１２ビット幅の命令
・１０２４×１２ビットの内蔵プログラムメモリ
・８ビット幅のデータバス
・６４×８ビットのレジスタファイル
　　内、３２本はシステム予約済み
・７レベルのプログラムカウンタ用のハードウェアスタック
・割り込み要求に対するバッファ
・４つの PWM
・４つのタイマ
・３０のシングルワード命令
・全ての命令が１クロックで実行可能

図２－１に全体のブロック図を示す。

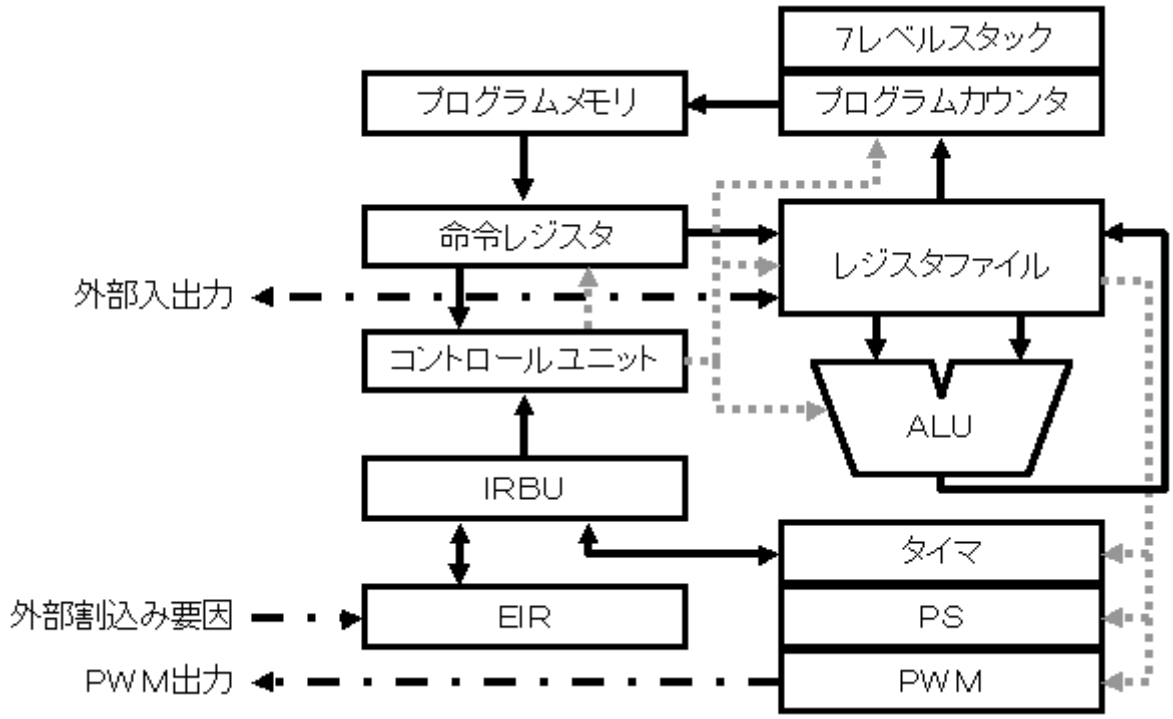

図２－１　ブロック図

２－１　命令

２－１－１　命令体系

命令語長を１２ビット、アドレスバスを１０ビット、データバスを８ビットとし、レジスタをシステム予約のものも含めて６４本用意した。これらハードウェア上の仕様の詳細は２－２節　実現方法で述べる。

命令セットはフォーマットによって大きく分けると、４種類に分類出来る。

・Ｔｙｐｅ１　ALU 演算（ADD や OR、SRA など）

Op[11:6]	Op[5:0]
オペコード	レジスタ指定

上位６ビットで指定される命令で ALU を動作させる。ALU の入出力には、アキュムレータと下位６ビットで指定されるレジスタの値を入力とし、アキュムレータに値を戻すものと、下位６ビットで指定されるレジスタの値を入力とし、そのレジスタに値を戻すものがある。

・Ｔｙｐｅ２　ビットアクセス命令（BC、BS、BTZJ、BTNZJ）

Op[11:9]	Op[8:6]	Op[5:0]
オペコード	ビット指定	レジスタ指定

下位６ビットで指定されるレジスタの内の１ビットを操作する命令。どのビットを見るかは、命令の第８ビットから第６ビットまでで指定し、どの操作を行うのかを上位３ビットで指定する。

・Ｔｙｐｅ３　即値指定命令（JMP、CALL、LD）

Op[11:8]	Op[7:0]
オペコード	即値指定

上位４ビットでどの命令を実行するのかを指定し、下位８ビットでジャンプ先や呼び出すサブルーチンのアドレスの上位８ビット、もしくはレジスタに代入する値を指定する。

・Ｔｙｐｅ４　その他の命令（RETURN、RJMP など）

上の３つに含まれないもの。

２－１－２　命令の動作の詳細

ALU 演算命令

ADD	二入力の和
ADDC	二入力とキャリーの和
SUB	二入力の差
SUBB	二入力の差からボローを引く
AND	二入力の論理演算（AND）
OR	二入力の論理演算（OR）
XOR	二入力の論理演算（ＸＯＲ）
R2A	レジスタからアキュムレータへのデータ転送
INC	レジスタの値のインクリメント
DEC	レジスタの値のデクリメント
NOT	レジスタの値をビットごとに反転
SRA	レジスタの値を算術的に右へ１ビットシフト
SRL	レジスタの値を論理的に右へ１ビットシフト
SLA	レジスタの値を算術的に左へ１ビットシフト
CLR	レジスタの値を０にする
A2R	アキュムレータからレジスタへのデータ転送

シフト命令において、算術的に右へ１ビットシフトとは上位７ビットをそのまま右へ１ビットずらし、最上位ビットには元の最上位ビットを入れることであり、論理的に右へ１ビットシフトとは最上位ビットには０を入れるということである。また、算術的に左へ１ビットシフトは下位７ビットをそのまま左に１ビットだけずらし、最下位ビットには０を入れる。これは、右へ１ビットずらすという動作が値を二分の一に、左へ１ビットずらすことが値を二倍にすることからきている。

また、シフト命令でレジスタから追い出されるビットはキャリーフラグに入れられる。

ALU 演算の前半の命令が結果をアキュムレータに入れるのに対し、後半の命令は結果を入力として指定されたレジスタに戻す。これを利用したのが R2A と A2R で、R2A は入力されたレジスタの値をアキュムレータに、A2R はアキュムレータの値を指定されたレジスタに代入し、データの転送を行う。

ビットアクセス命令

BC	指定したビットの値を０にする
BS	指定したビットの値を１にする
BTZJ	指定したビットの値が０ならジャンプ
BTNZJ	指定したビットの値が１ならジャンプ

即値指定命令

JMP	指定された値を上位８ビットのアドレスとしてジャンプ
CALL	指定された値を上位８ビットのアドレスとしてサブルーチンの呼び出し
LD	指定された値をレジスタに代入

その他の命令

RETURN	サブルーチンからの復帰
IRETURN	割り込みサブルーチンからの復帰
IDFC	割り込み禁止フラグを０にする
IDFS	割り込み禁止フラグを１にする
RJMP	現在の値から指定された値だけジャンプ
ELD	外部からの入力を任意のレジスタに代入
NOP	何もせずに次の命令に進む

２－２　実現方法

２－２－１　入出力の説明

入出力ポートには次のものがある。

入力

・ＣＬＫ（クロック）信号

クロック信号で、本 CPU はこれのエッジに同期して動作する。

・ＲＥＳＥＴ信号

リセット信号で、これがアサートされている間は全回路が初期状態になる。

・ＲＵＮ信号

プログラムの実行にはリセット信号がネゲートされ、このラン信号がアサートされている必要がある。リセット信号がネゲートされていても、ラン信号がネゲートされているとプログラムの実行はなされない。逆にラン信号がアサートされていても、リセット信号がアサートされていれば全回路が初期状態にリセットされ、プログラムの実行は行われない。

・外部割り込み信号

外部からの割り込みを受け付けるためのもので、４つある。

・外部からの入力

８ビット×２の外部からの入力を受けられる。

出力

・PWM 出力

可変長のパルスを４系統出力出来る。

・外部へのデータの出力

８ビット×２の外部への出力が出来る。

２－２－２　各モジュールの説明

今回のチップの作成に当たって、一つの大きなモジュールを作るのではなく、機能ごとに分けて作った小さなモジュールを集めて、目的とする機能を持った回路を構成した。

ここでは、それらの個々のモジュールについて簡単な説明をする。

１）算術論理演算ユニット(ALU)

二つの入力を受けて、算術演算もしくは論理演算を行い、結果を出力する。

二つの入力の内、一つはアキュムレータの値（もしくはマスクデータ)、一つはレジスタファイルの中の選択されたレジスタの値である。

ALU はマルチプレクサと外部からの割り込みを受け付ける EIR を除いて唯一クロックに非同期のユニットであり、クロックの立ち上がりの後で RU の出力が安定してから RU が値を取り込むクロックの立ち下がりまでに、出力が安定していなければならない。従って、今回の回路で最もパスが長く、ボトルネックになりやすい個所である。

また、演算実行時には結果の付属情報として、フラグを出力する。そのフラグには以下のようなものがある。

i)キャリーフラグ

加算時に最上位ビットにキャリー（桁上がり）があれば１に、なければ０にセットされ、減算時にボロー（借り）があれば１に、なければ０にセットされる。また、論理演算時には常に０にセットされる。

ii)オーバーフローフラグ

算術演算の結果が８ビットから溢れた場合、１にセットされる。

iii)パリティフラグ

パリティ（８ビットの中に含まれる１の数）のチェックを行う。偶パリティで０に、奇パリティで１にセットされる。

iv)ゼロフラグ

演算結果が０であれば１に、０でなければ０にセットされる。

２）レジスタファイルユニット(RU)

システム予約のものと、そうでないものがある。システム予約のものはアキュムレータやステータスフラグとして、またはプリスケーラやPWM、タイマなどのコントロールとして用いられる。システム予約でないものはデータを保持するために用いられるが、システム予約のものでも、あらかじめプログラムで使用されないことが分かっていれば、データの保持に用いることが出来る。

入力には、ALUの出力、ステータスフラグの値、外部からの入力を受け付ける。

また、システム予約のレジスタ用途を表２－１に示す。

００	アキュムレータ	０１	ステータスフラグ
０２	ジャンプ先アドレス	０３	データインデックス
０４	テンポラリレジスタ	０５	レジスタポインタ
０６	間接アドレス	０７	P.T.ctrl*1
０８	PWM０のLL*2	０９	PWM０のHL*2
１０	PWM１のLL*2	１１	PWM１のHL*2
１２	PWM２のLL*2	１３	PWM２のHL*2
１４	PWM３のLL*2	１５	PWM３のHL*2
１６	外部割り込み０*3	１７	外部割り込み１*3
１８	外部割り込み２*3	１９	外部割り込み３*3
２０	タイマ０による割り込み*3	２１	タイマ１による割り込み*3
２２	タイマ２による割り込み*3	２３	タイマ３による割り込み*3
２４	タイマ０の割り込み発生周期	２５	タイマ１の割り込み発生周期
２６	タイマ２の割り込み発生周期	２７	タイマ３の割り込み発生周期
２８	外部出力ポートA	２９	外部出力ポートB
３０	PWMのクロック分周比	３１	タイマのクロック分周比

*1　PWMとタイマのコントロール用

*2　LLはLow(0)を出す期間、HLはHigh(1)を出す期間

*3　割り込み発生時に呼び出されるサブルーチンの開始アドレス

表２－１　システム予約のレジスタの用途

以下にこれらのレジスタの詳細な説明をする。

i)reg00:アキュムレータ

アキュムレータはALU命令の二入力演算時に、入力の一方として、また計算結果を保持しておくために用いられる。

ii)reg01:ステータスフラグ

ステータスフラグはALU演算が行われた時に、ALUが出すキャリーなどの状態フラグの値を保存するために使われる。

ステータスフラグに格納されるフラグ情報を次の図２－２に示す。

7	6	5	4	3	2	1	0
S	0	0	P	ZC	C	O	Z

Sフラグ　　：サインフラグ、ALUの出力の最上位ビット
Pフラグ　　：パリティフラグ
ZCフラグ　　：ゼロフラグとキャリーフラグのORとったもの
Cフラグ　　：キャリーフラグ
Oフラグ　　：オーバーフローフラグ
Zフラグ　　：ゼロフラグ

図２－２

iii)reg02:ジャンプ先アドレス

ビットテスト後のジャンプの際に、ジャンプ先アドレスを指定するために使われる。ジャンプ先アドレスは実際のアドレスの上位８ビットを指定する。

iv)reg03:データインデックス

LD命令で与えられる８ビットの即値を、どのレジスタに代入するかを指定する。

v)reg04:テンポラリレジスタ

特に用途は指定されておらず、プログラムカウンタと同様のハードウェアスタックを持ち、サブルーチンの呼び出しによって値が変化しないので、一時的に値を置いておくために用いることが出来る。

vi)reg05:レジスタポインタ、reg06:間接アクセスレジスタ

レジスタ０６のポインタで、レジスタ０６にアクセスした時に実際にアクセスされるのは、レジスタポインタで指定されるレジスタである。但し、レジスタ０６自体は仮想的な存在で、レジスタポインタがレジスタ０６を示していても、レジスタ０６そのものに書き込むことは出来ない。

vii)reg07:P.T.ctrl

PWM とタイマのコントロールをする。下位４ビットが PWM の、上位４ビットがタイマの制御を行う。それぞれ下位の方から順に０、１、２、３と番号が割り振られている。ビットが１であれば動作させることになり、ビットが０であれば動作させないことになる。

viii)reg08 から reg15:PWM のパルス幅制御

PWM のパルスの幅を与えるために用いられる。LL は０を出力する期間であり、HL は１を出力する期間である。これらを足したものが PWM の出力周期になる。また、動的に周期を変えることも可能である。

LL と HL の最小単位はプリスケーラによって定義される。

ix)reg16 から reg23:割り込み発生時のサブルーチン呼び出し

割り込みが発生した時に呼び出されるサブルーチンの開始アドレスの上位８ビットを与えるために用いられる。

x)reg24 から reg27:タイマの割り込み発生周期

タイマが割り込みを発生させる間隔を設定する。タイマの最小単位もプリスケーラによって定義される。

xi)reg28 と reg29:外部への出力ポート

ここに書き込まれた値が外部へ出力されることになる。

xii)reg30:PWM のプリスケーラ、reg31:タイマのプリスケーラ

PWM とタイマの最小時間間隔を設定する。これらに代入されている値の数だけクロックが来ると、PWM やタイマにインクリメント信号を出力し、PWM やタイマはその信号を基準にして動作する。

３）インストラクションユニット(IU)

プログラムカウンタやスタックポインタ、命令レジスタを管理する。
入力として、ジャンプ先のアドレスやサブルーチンの開始アドレス、そしてプログラムメモリからの命令データを受け付ける。

プログラムカウンタとスタックポインタはクロックの立ち下がりエッジで動作し、命令レジスタはクロックの立ち上がりで動作する。プログラムカウンタの値が確定して初めてプログラムメモリから正しい次の命令データが来るので、メモリはクロック周期の半分の時間で動作しなければならない。

プログラムカウンタは図２－３に示すように配列の様になっており、スタックポインタによって一つのプログラムカウンタが選ばれる。スタックポインタは、サブルーチンが呼ばれるとインクリメントされ、サブルーチンからの復帰時にデクリメントされるため、サブルーチンを呼び出した時のプログラムカウンタの値が保持されることになる。

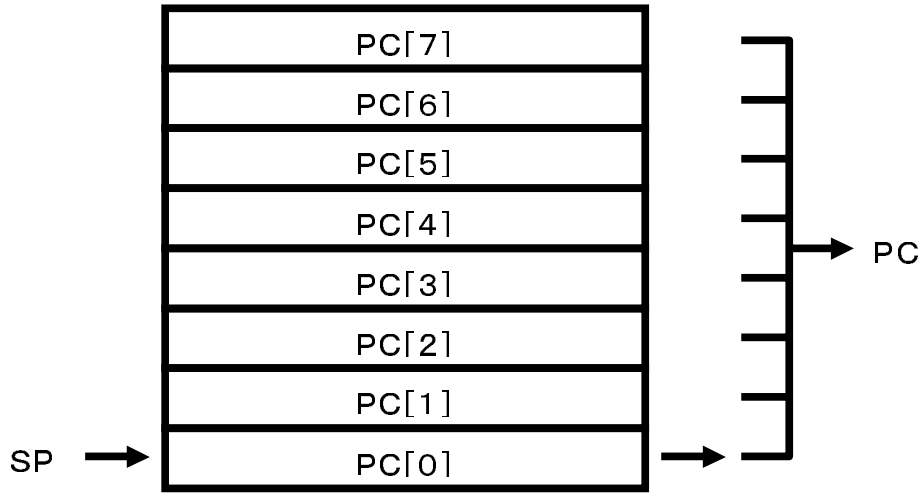

図２－３　インストラクションユニット

システム予約済みのレジスタの内、アキュムレータ、ステータスフラグ、ジャンプ先アドレス、データインデックス、テンポラリレジスタ、レジスタポインタはIUのプログラムカウンタと同様にハードウェアスタックを持ち、スタックポインタが共通であるため、サブルーチンの呼び出し時に自動的に値がスタックへ待避される。

４）コントロールユニット(CU)

１～３で紹介した３つのユニットを制御するユニット。命令をデコードし、割り込みがないか監視しながら、上記３つのユニットに対する動作と入力の選択を行う。

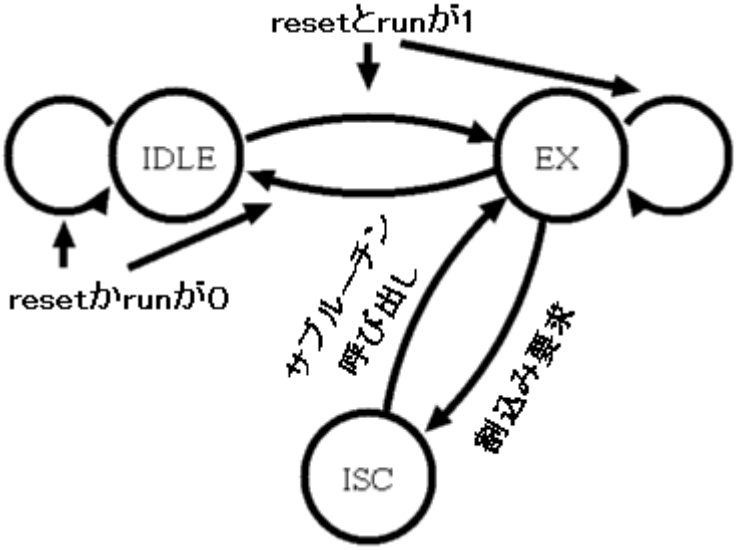

図２－４　状態遷移図

コントロールユニットは内部に IDLE、EX(Execute)、ISC(Interrupt Subroutine Call)の３つの状態を持ち、図２－４に示すようになっている。リセットがかかっている間は IDLE 状態であり、リセットとラン信号がアサートされると内部状態が EX に遷移してプログラムの実行が始まる。プログラムの実行中は基本的に内部状態は EX であり、割り込みが発生すると ISC へ状態遷移が起こり、サブルーチンが呼び出される。

リセット信号を０にすると、全ての内部状態は初期状態（全てのレジスタの値が０）になるが、RUN 信号が０でリセット信号が１の時は内部状態を保ったまま、プログラムの実行を止めることが出来る。

５）プリスケーリングユニット(PS)

クロックの分周を行い、PWM やタイマのプリスケーラとして働く。このモジュールの制御はレジスタファイル中のシステム予約済みのレジスタに保持されている値によって制御される。

その動作の例を図２－５に示す。この例ではプリスケーラのクロック分周比は３である。

６）パルス幅変調ユニット(PWM)

レジスタファイルに保持されている値を元に、幅を変調したパルスを出力する。

実際には、プリスケーラから来るインクリメント信号が LL 回来るまで Low(0)を出力し、次にインクリメント信号が HL 回来るまで High(1)を出力し、パルス幅の制御を行う。

その動作の例を図２－５に示す。この例では PWM の LL は２、HL は３である。

７）タイマユニット(Timer)

レジスタファイルに保持されている値を周期として、割り込み要求を出す。

PWM と同様に、プリスケーラから来るインクリメント信号がレジスタ２４から２７で指定されている回数だけ来た時に、割り込み信号を出力する。

その動作の例を図２－５に示す。この例ではタイマの周期は３である。

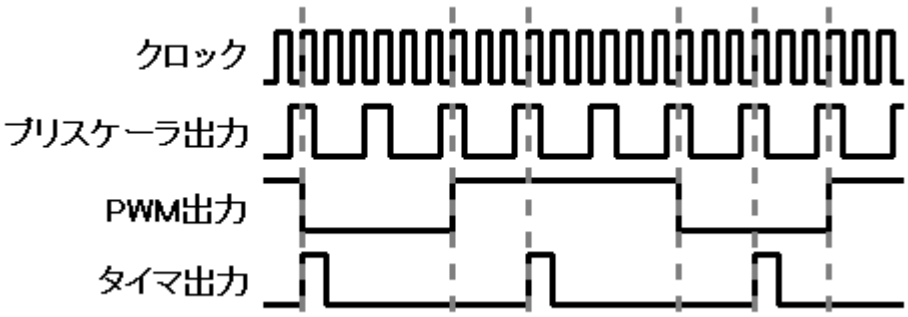

図２－５　プリスケーラ、PWM、タイマの動作例

８）外部割り込み受領ユニット(EIR)

外部から来た割り込み要求（非同期信号）を次に示すIRBUが受け取れる形（クロックに同期した信号）にする。

割り込み要求のパルスが入力されると、EIRユニットはIRBUに対して登録信号を出力する。IRBUは割り込みを感知して、それをバッファに入れるとクロックの立ち上がりに同期して、応答信号を出す。IRBUからの応答信号を受けると、クロックの立ち下がりで登録信号を立ち下げる。

その動作例を図２－６に示す。

９）割り込み要求バッファユニット(IRBU)

外部からとタイマからの割り込み要求を受け付け、順番待ちを制御する。

仮に複数の割り込みが同時に発生したとしても、全ての割り込み要因は優先順序による調停を受けて、それぞれに割り当てられた識別番号が一つずつバッファに入れられていく。一つバッファに入れられるごとに、書き込み用のポインタがインクリメントされるので、全ての割り込み要因がバッファに登録されることになる。

その動作例を図２－６に示す。

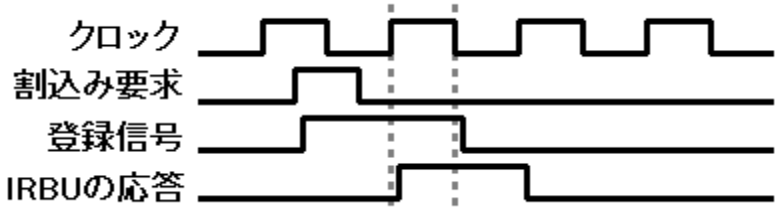

図２－６　EIRとIRBUの動作の例

２－２－３　命令動作時のフロー

この節では、各モジュールがどのように動作して各命令を実行していくのかについて説明するが、個別の動作の説明に入る前に、プログラム実行時の基本的な動作について述べる。

プログラムの実行に際して、プログラムカウンタの値がまず確定し、クロックの立ち上がりエッジの前にメモリの値が命令レジスタに入力される。そして、クロックの立ち上がりエッジで命令が命令レジスタに取り込まれ、コントロールユニットがデコードを開始する。すると、クロックの立ち下がりエッジに同期して、各モジュールはコントロールユニットの出す信号に従って動作し、命令が実行される。プログラムが実行されている間は、ジャンプ命令やサブルーチンコールで突然変わらない限り、プログラムカウンタの値はクロックの立ち下がりエッジでインクリメントされる。

これらのコントロールユニットとインストラクションユニットの動作は基本的には共通なものであり、以下の説明でも特に触れない限りは、ここに示した動作を行うものとする。

１）ALU 演算命令

ALU 演算では、主に ALU と RU が動作する。

まずクロックの立ち上がり時に、RU からはアキュムレータの値が出力され、ALU に送られる。また、ALU のもう一方の入力として、RU からの出力の内命令で選ばれたレジスタの値が入れられる。そして、ALU が二入力を処理して RU に結果を入力する。

次に、クロックの立ち下がりで RU が ALU の出力結果をレジスタ（アキュムレータ、もしくは命令で指定されたレジスタ）に代入し、ALU 演算命令が実行される。

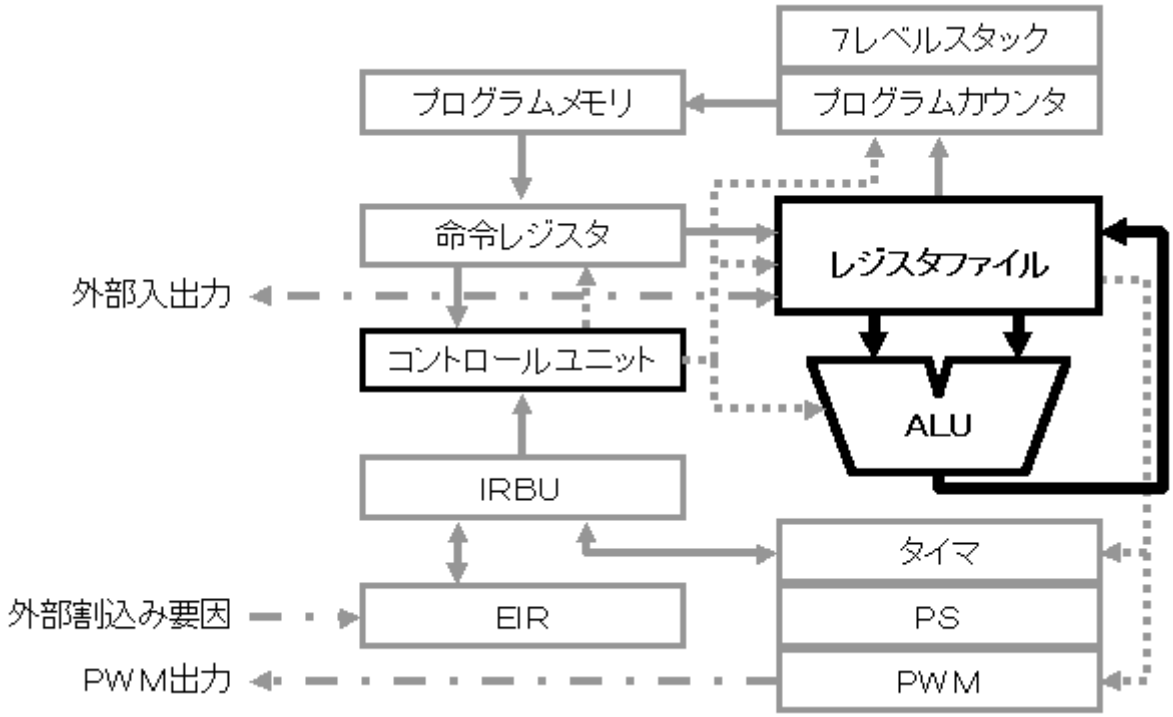

図２－７　ALU 演算命令、ビットアクセス命令時の流れ

２）ビットアクセス命令(BS、BS)

ビットアクセス命令では、ALU と RU が動作する。

ALU 演算と同様に、クロックの立ち上がり時に RU から命令で選ばれたレジスタの値が ALU に入力される。ALU 演算と違うのは、ALU のもう一方の出力がアキュムレータからのものではなくて、マスクデータである点である。例えば、あるレジスタの３ビット目を０にしたい場合は、マスクデータとして１１１１１０１１が ALU に入力され、論理演算 AND が施される。また、逆に３ビット目を１にしたい時は、００００１００がマスクデータとして ALU に入力され、論理演算の OR が施される。

クロックの立ち下がりでは、ALU 演算と同様に RU が結果をレジスタに格納する。

３）ビットテストジャンプ命令(BTZJ、BTNZJ)

ビットテストジャンプ命令では、ALU と RU、IU が動作する。

動作はビットアクセス命令に似ており、クロックの立ち上がりで RU から命令で選ばれたレジスタの値が ALU に入力される。また、マスクデータとしてビットセット命令(BS)と同じ１ビットだけが１である値がが ALU のもう一方の入力とされ、論理演算の AND が施される。

クロックの立ち下がりエッジでは、ALU の出力の内、ゼロフラグの値によって IU が動作する。BTZJ の場合、ゼロフラグが０の時プログラムカウンタはインクリメントされ、１の時はレジスタ０２の値を上位８ビットとして下位に００を付け加えた全１０ビットの値が代入される。逆に、BTNZJ の場合、ゼロフラグが０の時はプログラムカウンタにレジスタ０２に００を付け加えて１０ビットにした値が代入され、１の時はプログラムカウンタの値がインクリメントされる。

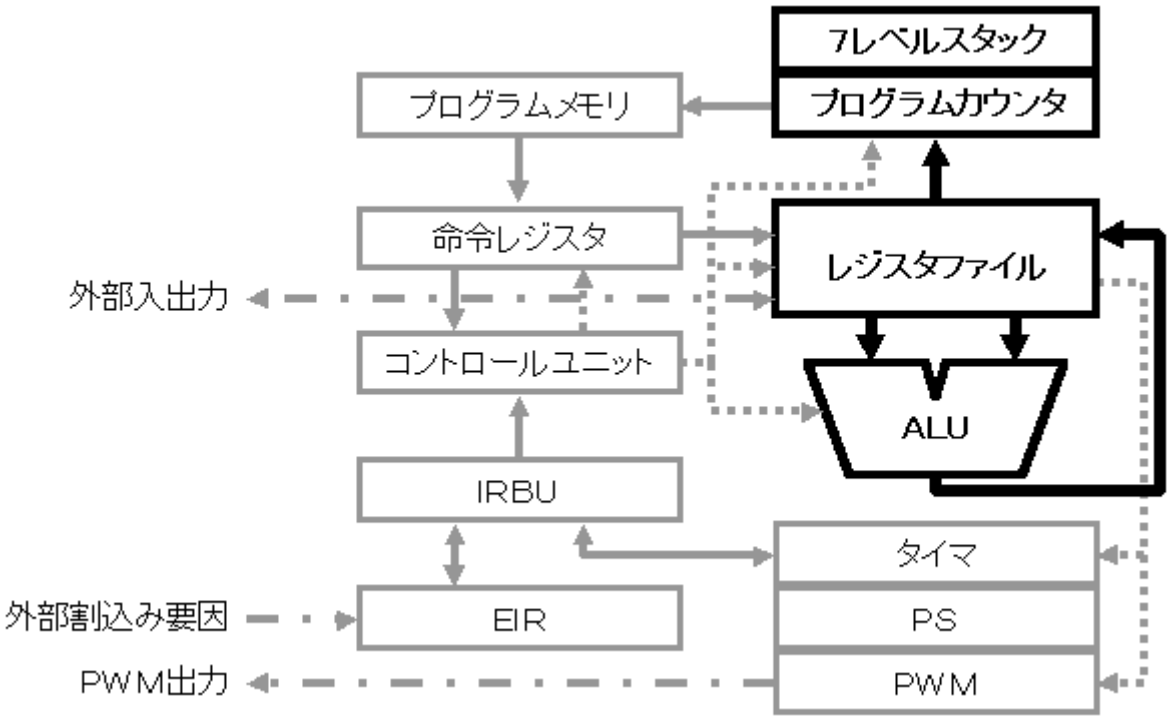

図２－８　ビットテストジャンプ命令時の動き

４）ジャンプ命令

ジャンプ命令で使われるモジュールは、IU である。

クロックの立ち下がりで、命令で直接指定された８ビットの値を上位に置き、下位に００の２ビットのデータを付けて１０ビットにしたアドレスをプログラムカウンタに代入する。

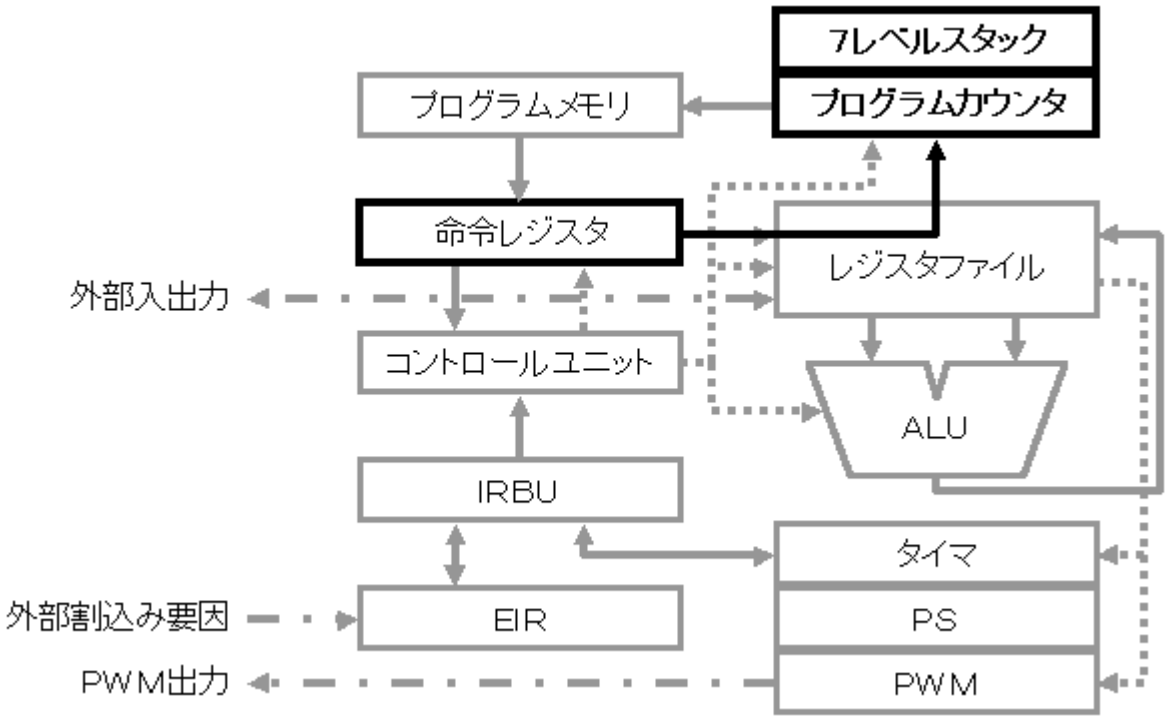

図２－９　ジャンプ命令、サブルーチンコール命令時の動き

５）サブルーチンコール命令

サブルーチンコール命令で使われるのは、IU である。

クロックの立ち下がりで、まずスタックポインタがインクリメントされ、プログラムカウンタを切り替えた後、ジャンプ命令同様、命令で指定された８ビットを上位に置き、下位に２ビットの００を付けて１０ビットに補完した値をサブルーチンの開始アドレスとしてプログラムカウンタに代入する。

割り込み発生時のサブルーチンの呼び出しは、このサブルーチンコール命令と基本的には同じであるが、CU が内部に持っている割り込み禁止フラグを１にセットする所が違っている。

6）ロード命令

ロード命令はRUだけが使われる。

クロックの立ち下がりで、命令で指定された８ビットの値が、レジスタファイル中のレジスタ０３の値で指定されるレジスタに代入される。

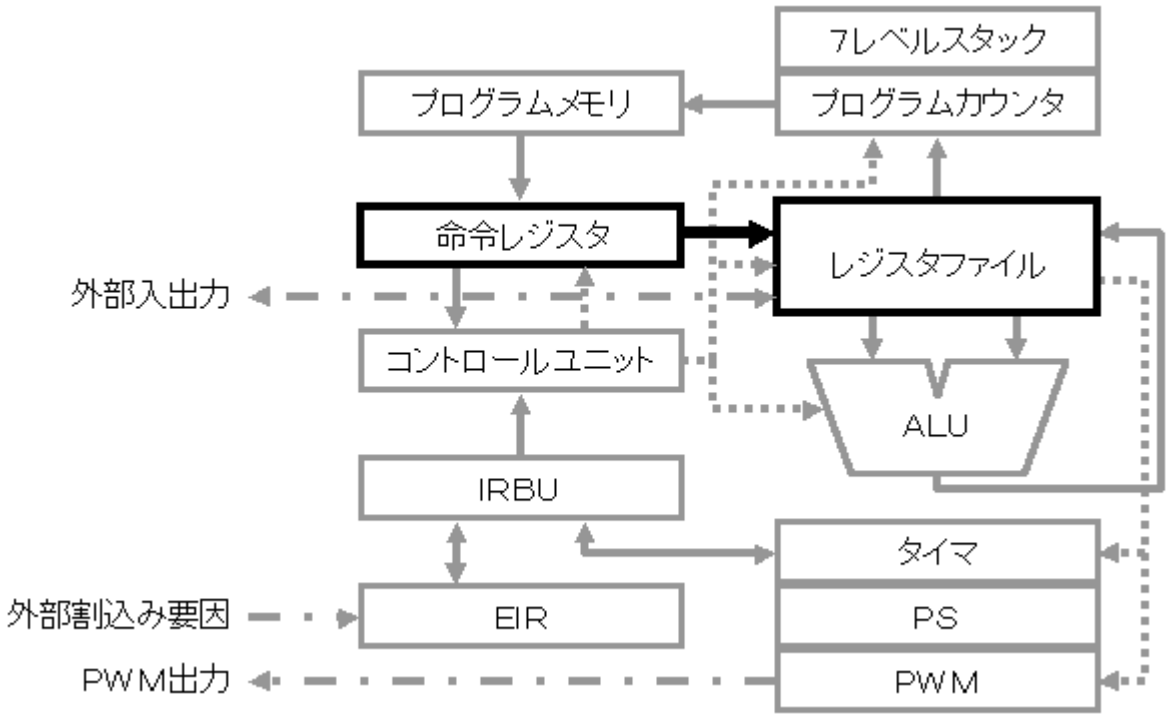

図２－１０　ロード命令時の流れ

7）サブルーチンからの復帰命令

サブルーチンからの復帰命令には、IUとCUが用いられる。

単なる復帰命令(RETURN)では、クロックの立ち下がり時にスタックポインタがデクリメントされるだけである。

割り込みによるサブルーチンコールからの復帰命令(IRETURN)では、クロックの立ち下がりの際にスタックポインタがデクリメントされるのと同時に、割り込み禁止フラグが０にリセットされる。

8）割り込み禁止フラグのセット／リセット

割り込み禁止フラグに関する命令は、CUだけが動作する。

割り込み禁止フラグのセットでは、割り込み禁止フラグを１に、リセットではフラグ

を０にする。動作のタイミングはクロックの立ち下がりである。

９）相対値ジャンプ命令

相対値ジャンプ命令では、IU が動作する。
クロックの立ち上がりで、プログラムカウンタの値と命令の下位６ビットを最上位ビットで補完（上位を最上位ビットで埋めて１０ビットにする）した値を加算し、クロックの立ち下がりでその値をプログラムカウンタに代入する。

１０）外部からのロード命令

外部からのロード命令では、RU が動作する。
クロックの立ち下がりで、外部からの入力を命令で指定されたレジスタに代入する。外部からの入力は８ビットのものが２系統あるが、どちらの入力の値を代入するかは、命令で指定される。

１１）NOP(No Operation)

ノップでは、デフォルトの IU（プログラムカウンタのインクリメント）と CU（デコード）の動作しかしない。プログラムの隙間埋め、もしくは時間稼ぎに用いられる。

第３章　開発環境について

開発環境としては、アセンブリを用意した。シミュレーション用のプログラムはこのアセンブラを用いて作られている。

アセンブラの機能としては、擬似命令が使えないので一般的なアセンブラには劣るが、ジャンプ先アドレスの指定やサブルーチンの開始アドレスの指定に用いる、行に対するラベル付けという最低限の機能だけは備えている。

以下に作成したアセンブラの書式を示す。但し、Function は命令を、Register はレジスタを、Label はジャンプ先のアドレスもしくはサブルーチンの開始アドレスを示すものである。

１）ALU 演算命令

[Label:]　Function　Register

Function で命令を指定し、Register で演算の対象となるレジスタを選択する。

２）ビットアクセス命令

[Label:]　Function　0-7,Register

Function で命令を指定し、０から７の値で Register の対象となるビットを決定する。

３）即値指定命令

[Label:]　Function　00-ff or Label

Function で命令を指定し、００～ｆｆの値か Label でジャンプ先のアドレスか、サブルーチンの開始アドレス、レジスタに代入する値のいずれかをを指定する。

４）その他の命令

i)サブルーチンからの復帰

[Label:]　RETURN

[Label:]　IRETURN

引数はなく、RETURN はプログラムカウンタをサブルーチンが呼び出される前の値に戻し、IRETURN は RETURN の機能とともに、割り込み禁止フラグを０にする。

ii)割り込みの許可／不許可

[Label:]　IDFC

[Label:]　IDFS

引数はなく、IDFC は割り込み禁止フラグを０にし、IDFS は割り込み禁止フラグを１にする。

iii)相対値ジャンプ

[Label:]　RJMP　00-3f

プログラムカウンタの値に、引数で指定された値を足し算する。但し、20-3f までは引き算になる。

iv)外部からの入力

[Label:]　ELD　A or B　Register

外部からの入力を Register で指定されるレジスタに代入する。外部からの入力は二系統あるので、どちらの入力を受け付けるかを A | B で選択する。

v)ノーオペレーション

[Label:]　NOP

引数はなく、プログラムカウンタの値をインクリメントする以外は何も実行しない。

第４章　チップへの実装

４－１　回路記述とシミュレーション

４－１－１　VerilogHDL による回路記述

目的とする機能を持った回路を VerilogHDL(Hardware Description Language)により記述した。

ハードウェア記述言語(HDL)を用いる利点としては、まず HDL 記述それ自体が設計の仕様書となることが挙げられる。また、シミュレーションを容易に行うことが出来るので、設計の誤りを早い段階で見つけることが出来ることも挙げられる。

４－１－２　Verilog-XL によるシミュレーション

VerilogHDL による記述の後、動作の確認のためシミュレーションを行った。なお、テストプログラムの作成には今回作成したアセンブラを用いた。

シミュレーションはテストスティミュラスと呼ばれる、外部からの刺激（入力）を与えることで行われる。様々な入力を与え、返ってくる出力を監視することで、欲する機能が得られたかどうかを確認する。

実際には、作成したモジュールとテストスティミュラスのファイルを Verilog-XL に読み込ませ、Simwave で入出力波形の観測をした。

４－２　論理合成とシミュレーション

４－２－１　Design-Analyzer による論理合成

Verilog-XL によるシミュレーションの終了後、次に Design-Analyzer による論理合成を行った。

論理合成とは、高位レベルのデザイン記述から、ライブラリコンポーネントに自動的にマッピングして下位レベルのデザインを実現するプロセスのことであり、ここでいうHDL の論理合成とは RTL(Register Transfer Level)で記述された設計をゲートレベル記述に変換するプロセスとする。

論理合成の長所として、トップダウン方式の設計ではシミュレーションを行うため、HDL 記述からゲート記述を得るまでの時間を短縮出来る、テクノロジによらない、シミュレーションが容易に出来ることが挙げられる。

４－２－２　Verilog-XL によるシミュレーション

論理合成後は、再び Verilog-XL によるシミュレーションを行った。

論理合成を行う前の、ゲート遅延を含まない理想的な条件下でのシミュレーションとは違い、実際に用いられるゲート回路の組み合わせで回路を実現したことで得られる、ゲート遅延を含んだより現実に近いシミュレーションを行うことが出来る。

４－３　配置配線とデザインルールチェック

４－３－１　Apollo による配置配線

論理合成によって出力されるネットリスト（ゲートとそれらを繋ぐ配線の情報）から、実際のセルの配置とそれに対する配線を、プロセスはロームの CMOS0.6μm ルールで、ライブラリは東京大学の EXD ライブラリを用いて、Milkyway と Apollo によって行った。以下にその工程を簡単に示す。

１）配置配線に先立って、まず Milkyway によって作業場所としてのライブラリを作成し、Apollo で用いられる形式に変換するため、ネットリストを読み込み、階層展開を行う

２）Apollo を起動し、１）で作成したライブラリを開き、作業を行うためのセルを作成する

３）２）で作成したセルに１）で階層展開したネットリストをバインドし、電源とグランドの接続を行う

４）フロアプランを行うことで、スタンダードセルを配置するコアの大きさや周囲の間隔を決定し、ロウの配置を行う

５）パッドを挿入し、電源リングを作成する

６）内側のスタンダードセルの動作が外側のものに遅れないようにするため、コアに電源とグランドのストラップを入れる

７）スタンダードセルの配置を行い、配線を考慮して並び替えを行う

８）スタンダードセルの電源とグランドの接続を行う

９）スタンダードセルの隙間のウェルを埋める

１０）大まかな配線を行い、長い配線を最短距離で繋げられるようにする

１１）詳細な配線を行い、デザインルールからのバイオレーションがなくなるように配線の改善を行う

１２）配線の最適化を行う

１３）LVS と DRC でデザインルールチェックを行う

以上のような工程で得られたレイアウトを図４－１に示す。

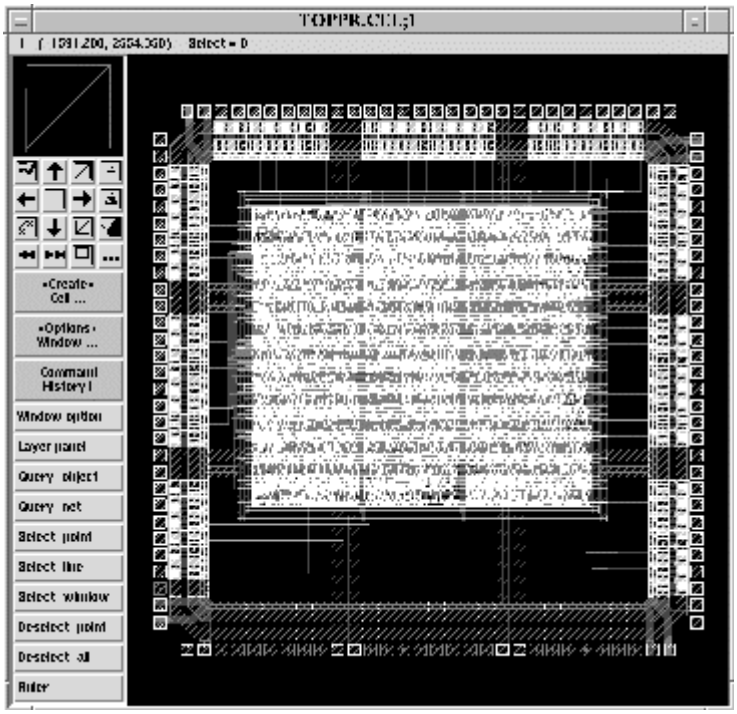

図４－１　レイアウト

第５章　まとめ

仕様の設計に始まり、HDL による記述、論理合成、シミュレーションを行い、期待通りの結果が得られたので、配置配線を行い、レイアウトを得た。また、アセンブラの作成も行い、シミュレーション用のテストプログラムの作成に用いた。

設計の際に問題点として挙げられるのは、ハードウェアの作成にしても、ソフトウェアの作成にしても、どちらか一方だけの事を考えていたり、どちらか一方だけを熟知しているだけでは良いものは作れないということである。

どんなに優れた機能を持ったハードウェアでも、使うことの出来る人がいなければ、どんなに優れたソフトウェアでも、それを実行出来るハードウェアがなければ意味がない。一見、当たり前のことのように聞こえるかもしれないが、重要なことである。

最後に、今回作成したチップのレイアウトデータは、VDEC(VLSI Design and Education Center)のチップ試作サービスにより作成を行う予定である。

謝辞

本研究を行うに当たって、指導教官として御指導頂きました鈴木正國教授に深く感謝致します。また研究面や私生活面で、御指導、御協力下さった北川章夫助教授、秋田純一助手にも心から感謝致します。私生活面や研究室行事などでお世話になった柿本芳雄技官に深く感謝致します。研究面で御協力下さった東京大学大規模集積システム設計教育研究センターさん、ロームさん、凸版印刷さんに感謝致します。

研究及び学生生活において御指導、御協力下さった博士後期課程３回生のT.K.Chakraborty さん、博士前期課程２回生の田口和彦さん、夏目雅弘さん、前多和洋さん、水橋嘉章さん、博士前期課程１回生の小川明宏さん、高瀬信二さん、中橋憲彦さん、早川史人さんに深く感謝致します。最後に研究室、日常生活を大変有意義にして下さいました、今井豊さん、佐藤堅さん、房川実さん、藤田隼人さん、水野浩樹さん、村上崇さん、渡辺晃さんに深く感謝致します。

どうもありがとうございました。

参考文献

［1］Z-80 アセンブリプログラミング
廣松恒彦＝編　船山邦夫・関口英幸＝共著

［2］Verilog-HDL によるトップダウン設計
E.Sternheim/R.Singh/R.Madhavan/Y.Trivedi　著　　井上博史/鈴木隆　訳