平成 16 年度秋田工業高等専門学校卒業論文

『２足歩行ロボットの製作と制御』

平成 17 年 3 月

報告者　　機械工学科 5 年　12-28　　豊島広樹

編 15-168　佐藤　侑

指導教員　　木澤　悟

一章 二足歩行ロボットに関する研究

1－1　研究背景

近年,人間との協調・共存を目指したロボットの研究開発が盛んである.ロボットがそのような状況で作業を行う状況を想定したとき常に作業環境,作業方法などの情報が事前に十分に与えられるとは考えづらい.そのためロボットが環境や作業に適応しなければならない.

また少子高齢化社会に伴う労働人口の減少や要介業者の増加に備え,人間自身のメカニズムを早急に解明することも必要となってきている.

そこで本研究では人間のメカニズムを解析するべく人間はどのようにして行動を行っているか,また人間の動き方つまり,人の関節はどのように動いているかなどを解析するため,人間の足を想定したロボットを製作し,そのロボットに股関節（3 箇所）,膝間接,下跳躍間接,足首の片足で計 6 箇所にサーボモータを使用し二足歩行の設計や制御の妥当性を検討することが目的である.

第二章　実験装置システムの概要

２－１　マイコンボード（SH2―7045F）

二足歩行ロボットを実現させるため,タイマやA/D変換器といった機能を十分に有した制御用マイクロコンピュータが必要である.そこで本研究では日立製作所製 32 ビットマイコンである SH2-7045F を使用した.仕様については表 2-1 に示す.また実際の写真を Fig.2-1 に示した.さらに Fig2-2 に SH2-7045F のブロック図を示した.

表 2-1.CPU の仕様

CPU	HD64F7045F28 ○内部 32 ビット構成 ○汎用レジスタマシン －汎用レジスタ 32 ビット×16 本 －コントロールレジスタ 32 ビット×3 本 －システムレジスタ 32 ビット×4 本 ○RISC タイプの命令セット －命令長：16 ビット固定長によるコード効率の向上 －ロードストアアーキテクチャ －遅延分岐命令の採用で，分岐時のパイプラインの乱れを軽減 －C 言語指向の命令セット ○命令実行時間　1 命令／1 サイクル（28MHz） ○乗算器内蔵 ○パイプライン　5 段パイプライン方式
キャッシュメモリ	○1kB 命令キャッシュ ○命令コード及び PC 相対読み出し・データをキャッシング ○内蔵 RAM と兼用，キャッシュイネーブル時は内蔵 RAM のうち 2kB をアドレスアレイ・データアレイとして使用
割り込みコントローラ	○外部割り込み端子×9 本 ○外部割り込み要因 44 要因
その他	大容量 ROM・RAM・タイマ・A/D 変換器・I/O ポート・ シリアルコミュニケーションインターフェース等を内蔵

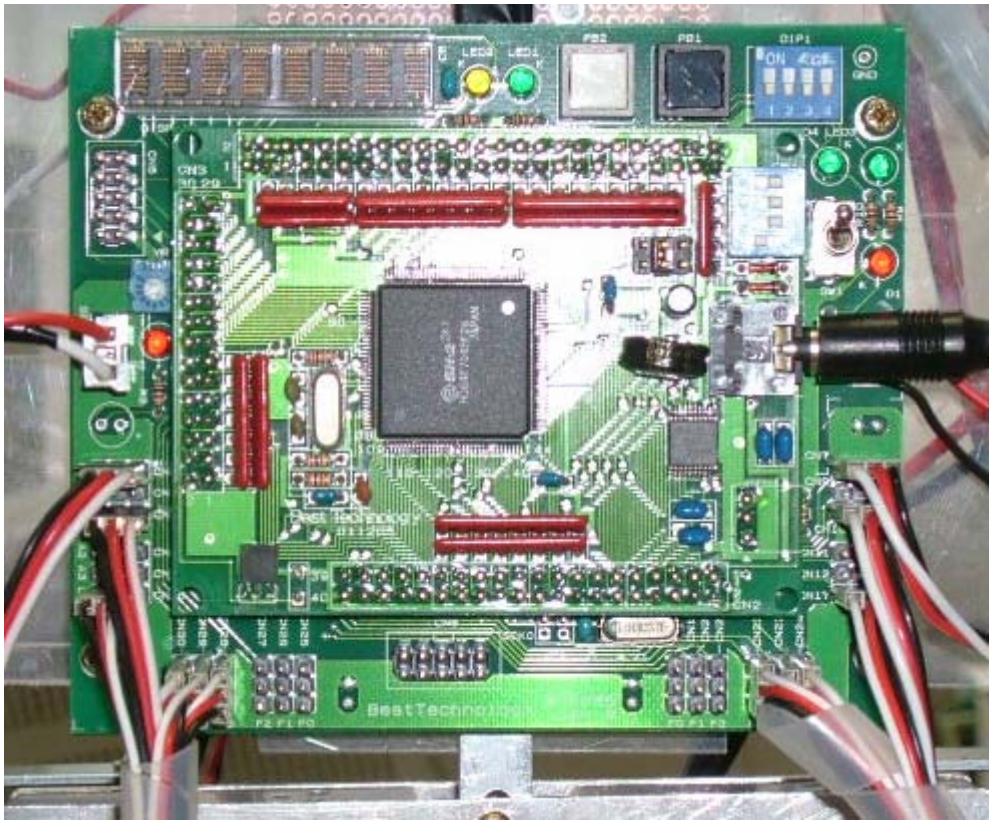


Fig.2-1 SH2－7045F

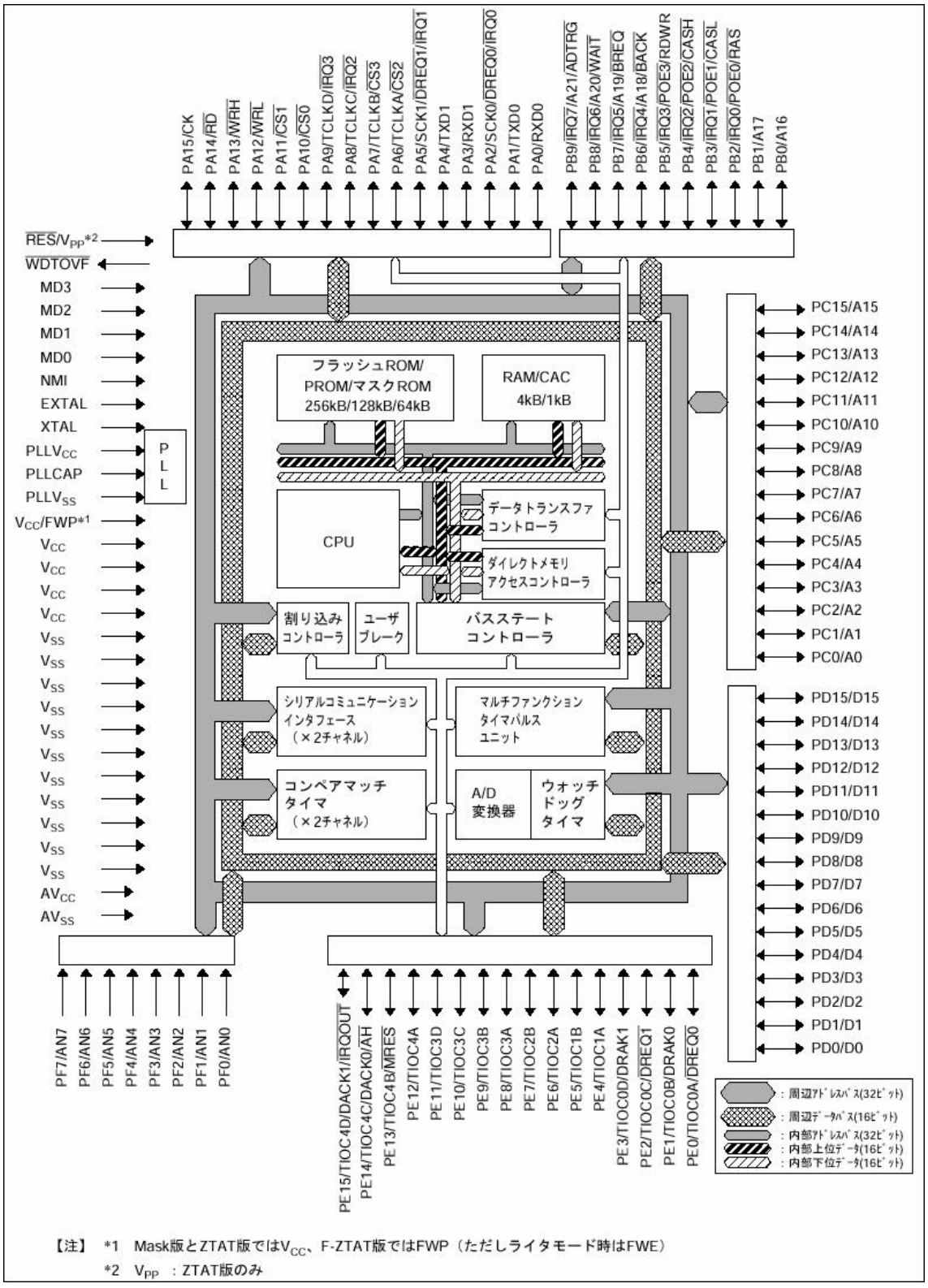


Fig.2-２ SH2-7045F のブロック図

２－２　開発環境

SH2-7045F マイコンボード用のプログラム開発環境としては，OS として Windows, 動作するパソコンが必要となる．ボードに付属されている統合開発環境ソフトウェア GCC Developer Lite【1】及びコンパイラ等ツールソフトウェア GNU Pro Toolkit【2】は Windows 上で動作し，これを用いることで，プログラムの編集，コンパイル，CPU への書き込みを行うことが可能である．
本研究では、コンピュータと SH2-7045F とを，Fig2-3 のように通信ケーブル(RC-232C)を用いて繋ぎ, コンピュータで C 言語を用いて作成したプログラムを SH2-7045F におくり,それらに対応するモータに送られたプログラムを実行することで制御している．
転送手順を表２－２に示した．

表 2-2 プログラム転送方法

１．ツールの SIMPLE　TERM をクリック．
２．マイコンボート上の「ディップスイッチ４」をオンにしたまま，マイコンボートの電源を供給する．(「ディップスイッチ４」をオフにしたままだと，以前のデータが保護されている状態にある．)
３．すると文字列が表示される．
４．マイコンボードの電源をオフにする．
５．「通信」の「ポート　オープン」をクリック
６．マイコンボート上の「ディップスイッチ４」をオンにしたままマイコンボートに電源を供給する．(制御電源を ON にする．)（マイコンボードは受信待機中になっている．)
７．「SIMPLE　TERM」の「転送」をクリック．(転送開始)

プログラム編集後のコンパイル手順
1.「ツール」の「GCC オプション」の順にクリック
2.「設定リストを SH2　７０４５F」増設 SRAM にして「OK」をクリック．
3. コンパイルする．

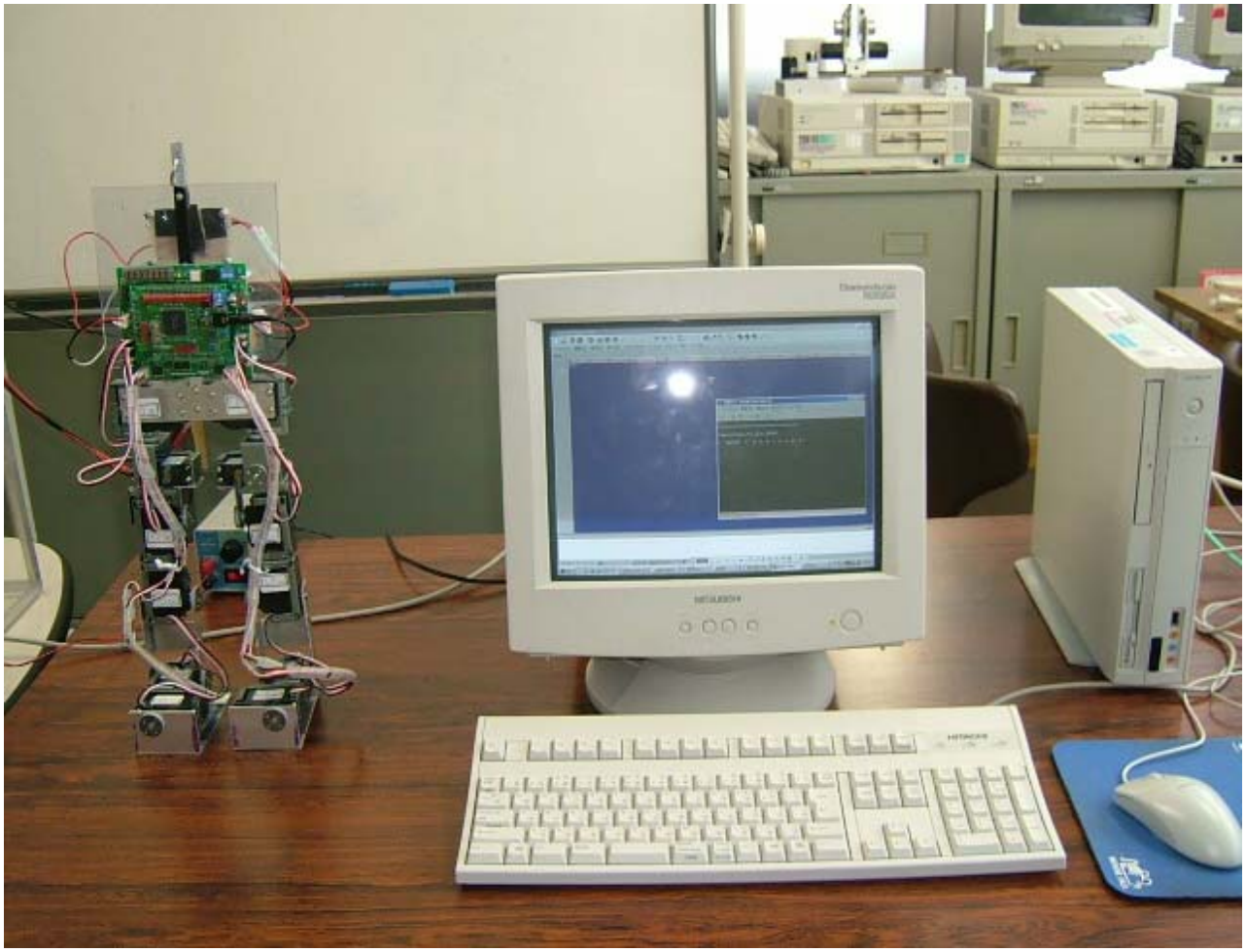


Fig.2-3　二足歩行ロボット開発環境

三章　二足歩行ロボット

３－１　組み立て図

本研究で作成した二足歩行ロボットは、厚さ 1.5mm のアルミ材を使用している.これはできるだけ、サーボモータへの負荷をできるだけ軽減するため比重を小さく単位重量あたりの強度があり、かつ低価格であるものを選んだ結果である.Fig.3-2 には組立図を示し、ロボットの脚部の写真を Fig.3-1 に示す.なおロボットの重量は 1.59kg である.

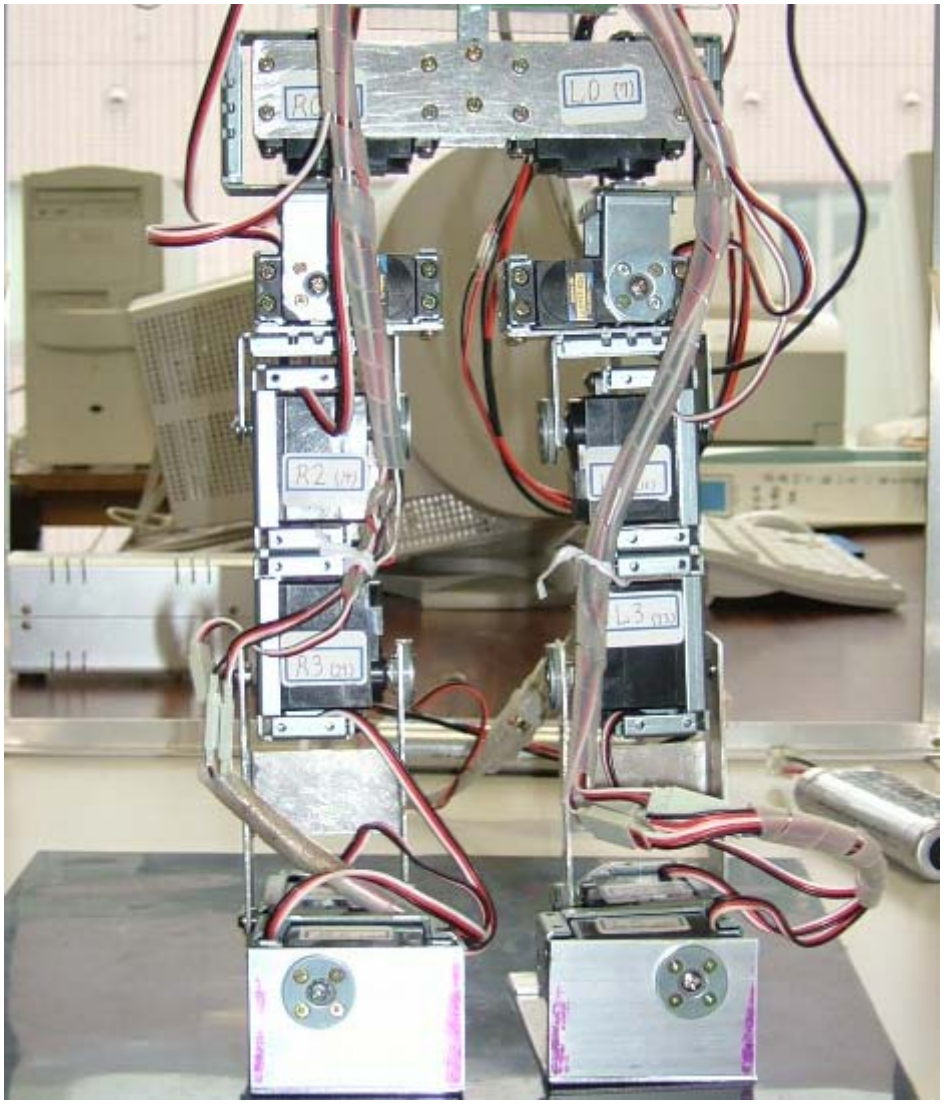


Fig3-1 二足歩行ロボット組み立て図

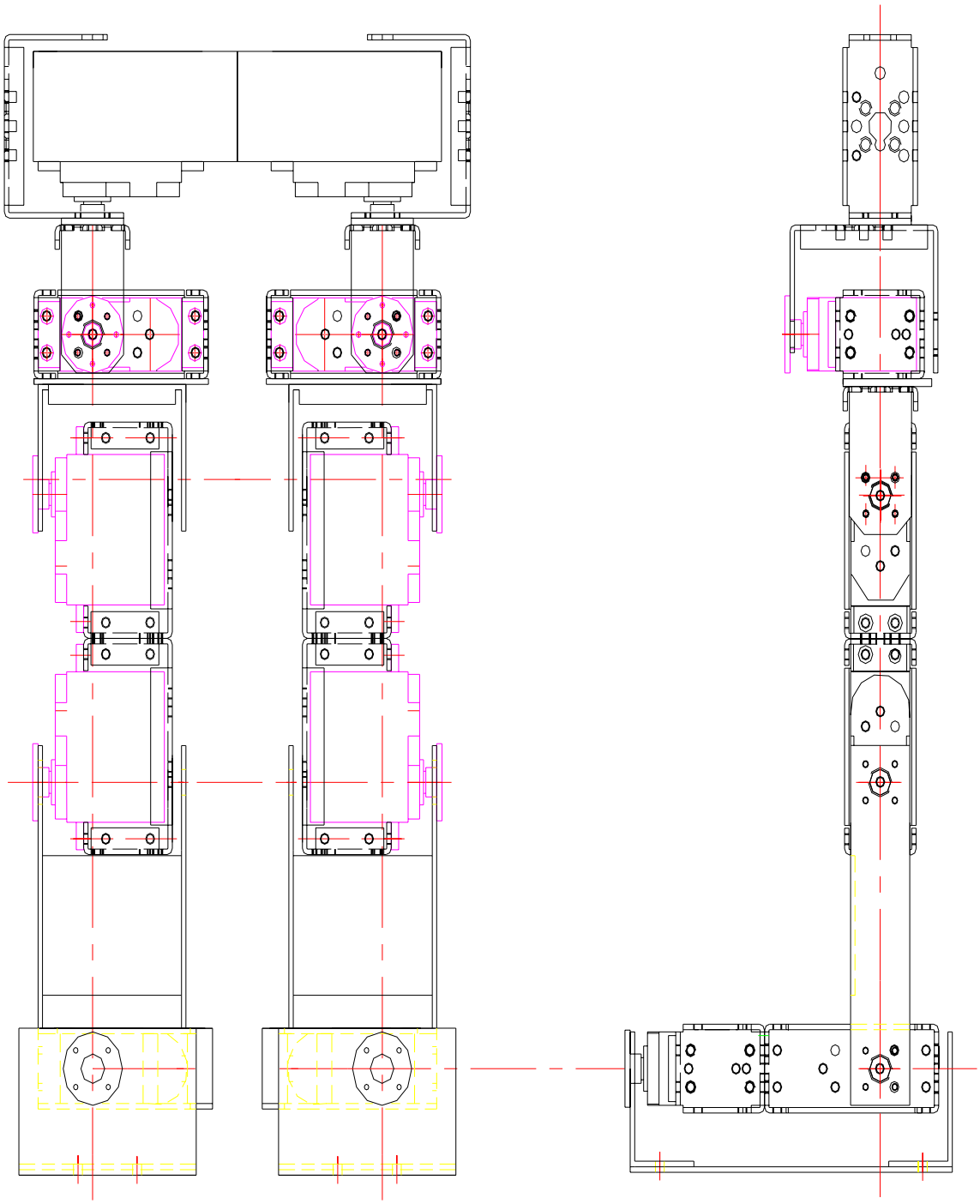


Fig3-2 二足歩行組み立て図

３－２　駆動部

今回は Fig3-1 からわかるように、駆動部が片足６個の合計１２カ所用いられている.
駆動部として用いられているサーボモータは、近藤科学社(株)の PDS－2344FET と PDS-2144FET を用いている.

３－２－１　サーボモータ

制御回路を含むモータモジュールである。この制御回路に、制御パルスを送ると、そのパルス幅に比例した角度にまで回転する。パルスの幅と回転角度の対応は、それぞれのサーボモータで、少しづつ違ってくる.

３－２－２　サーボモータの仕様

サーボモータの図と仕様を以下に示した.

Fig3-３ PDS-2344FET

表３－１　PDS-2344FET の仕様

スピード	0.13sec/60°
トルク	13.0kg・cm
重量	54.5g
寸法	41×38×２０

Fig3-４　PDS-2144FET

表３－２　PDS－２１４４FET の仕様

スピード	0.13sec/60°
トルク	13.0kg・cm
重量	54.5g
寸法	41×38×２０

３－３　サーボモータの制御

サーボモータを任意の角度まで回転させるためには，制御パルスのパルス幅を変調させることが必要となる．これを PWM（パルス幅変調）方式といい，サーボモータの制御に不可欠なものといえる．使用した CPU である SH2-7045F にはインターバルタイマ機能*が内蔵されているため，インターバルごとに任意のパルス幅をサーボモータに与えることで PWM 出力を行うことが可能であり，これを用いて制御を行う．なお，この制御は一方的に制御パルスを与えフィードバックは行わないオープン制御である．

サーボモータと SH2-7045F マイコンボードの接続基盤の構造は Fig3-5 のようになっている．また，I/O ボートのサーボモータ用のピン配列は Fig.3-6 の示す．なお，図中の S 表示はシングル，D 表示は，切り替えによるポートである．

(注＊インターバルタイマについては，４章にて説明．)

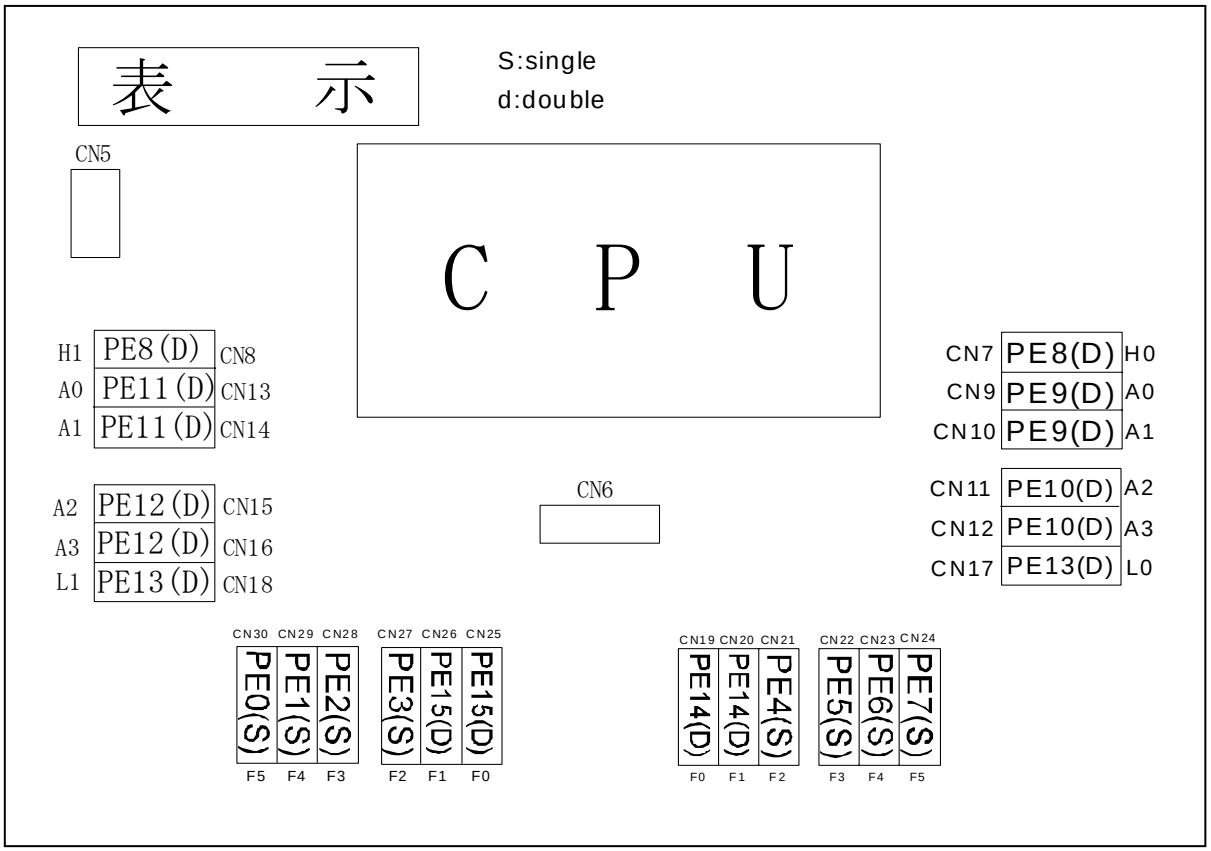


Fig.3-5　SH2―7045F マイコンボードの接続基盤の構造

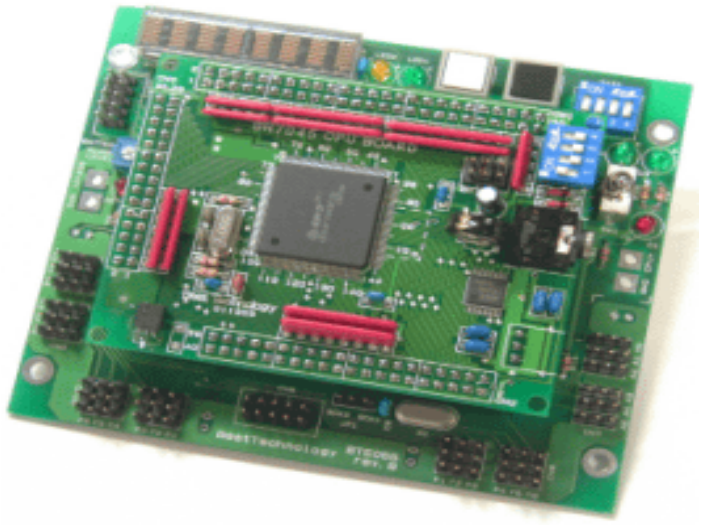


Fig3-6　I/O ボード

このボードでは，16 本の PWM しか使うことができないが，足のモータだけで，片足 6 本，両足で計 12 本の PWM を使うことになる。今は 16 あれば，十分足りるが，今後手を作ると 16 本だけでは，足りないということが考えられる．そこで，PWM を増やしてやる必要がある．PWM を増やす方法には次の 2 つがあげられる．

- I/O ポートで PWM を出力する方法
- PWM を切り替えて出力する方法

今回は PWM を切り替えて出力する方法を使用する．サーボモータの PWM については，Fig.3-7 のようになっている.図中のパルス幅は、1.48ms になっているが、これはサーボモータのニュートラル位置に対応している．

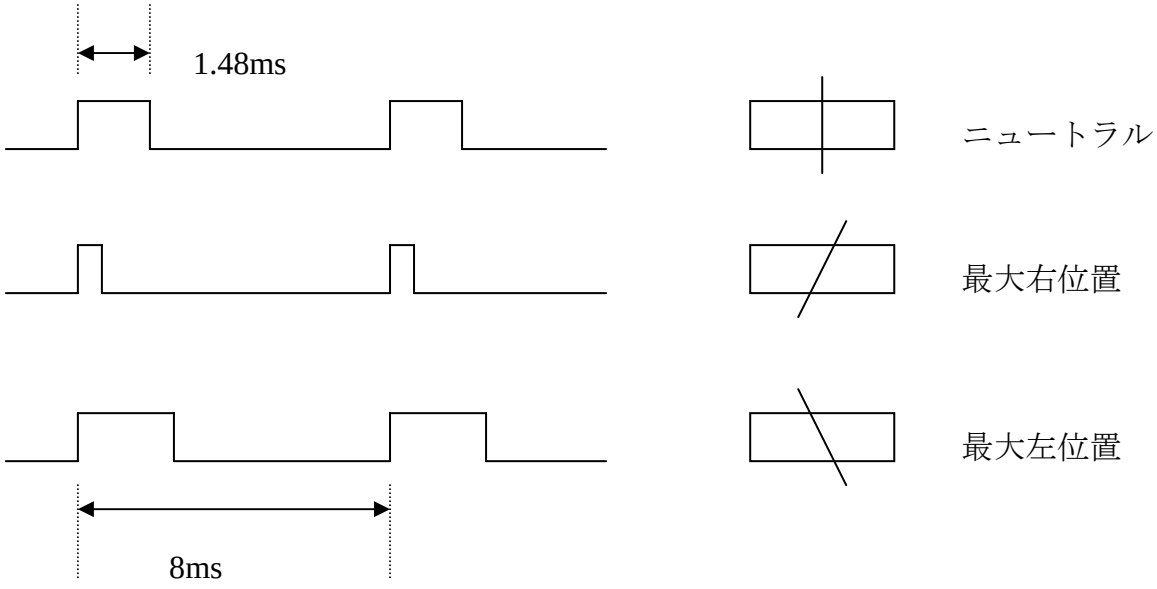


Fig.3-7　サーボモータの PWM

PWM の周期は一般に 10msec であり，サーボではこの周期を短くすることができる．今回は Fig.3-7 に示すように,この周期を 8ms に設定した.

次にサーボ出力のためのチャンネル数を増やす方法を述べる.Fig.3-8 に示すような時間的な割り込み技術を用いればハードウェア的な切り替え方法が使用できる．つまり，1ch の PWM 出力を２つに分けてチャンネル数を増加させようというものである.

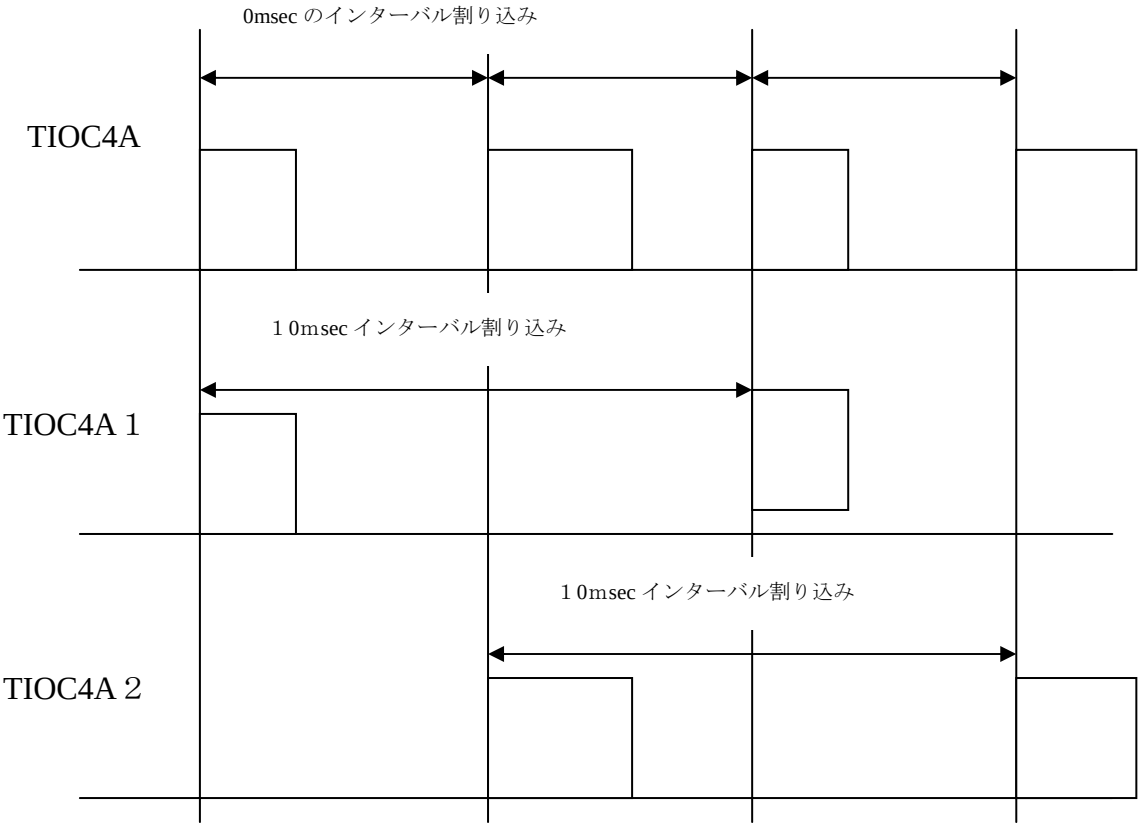


Fig.3-8　切り替え方法

Fig3-9 はチャンネルを切り替えるための回路図である．これには，AND の演算素子が使用されており．この回路を使用することでチャンネルを増加することができる. つまり, 図中の TIOC4A1 から PWM 信号を出力したい場合は, PA16 を high にすればよい. TIOC4A2 から PWM 信号を出力したい場合は PA17 を high にすればよく，TIOC4A ポートを切り替えることで，二つの出力を可能としている．本来であれば，16 本が使用可能であるが，このうち 8 本のチャンネルを，二倍とし，計２４個のチャンネルを使用することができる．また出力端子の構造を Fig3-10 に示す.

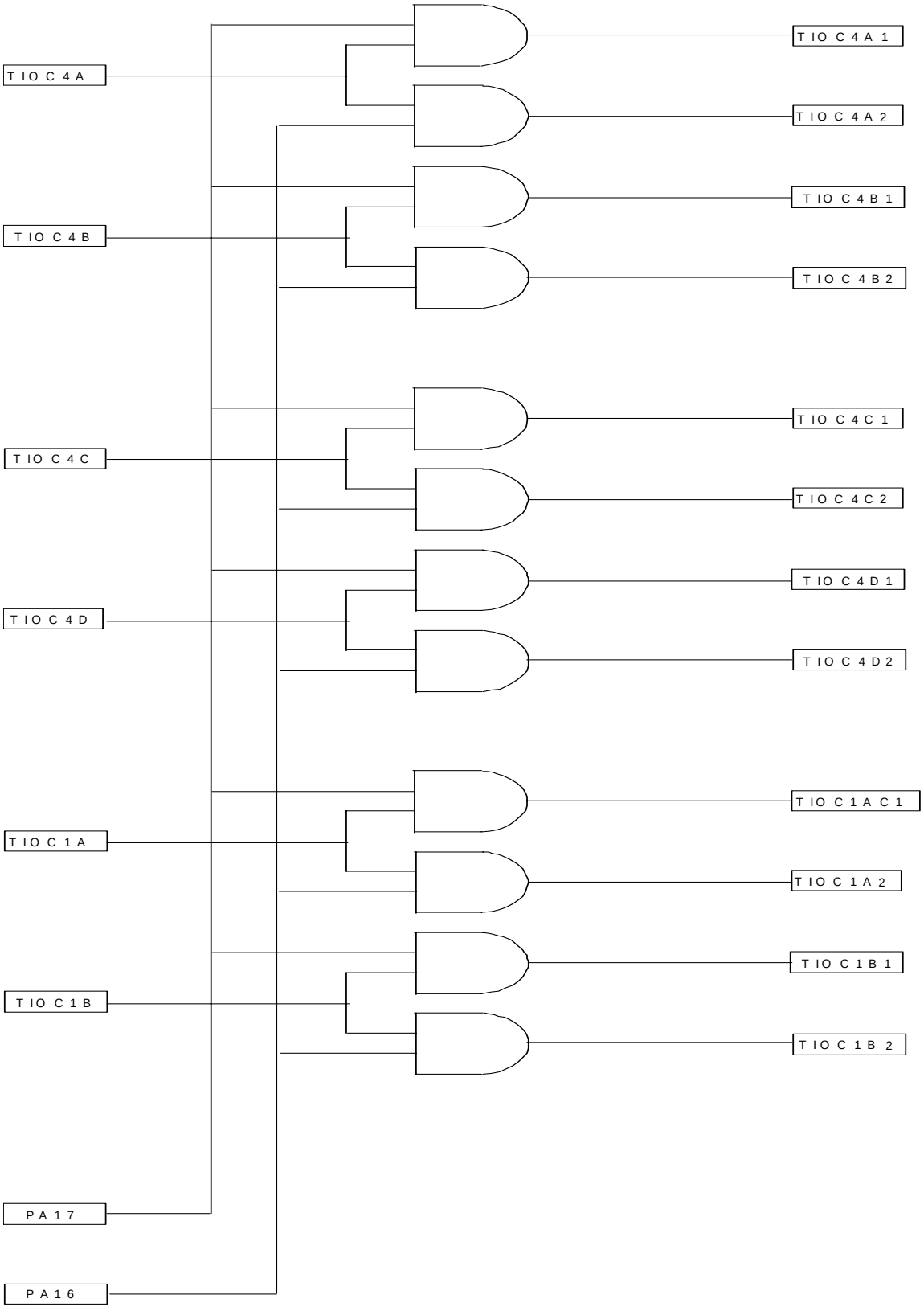


Fig.3-9　　切り替え回路図

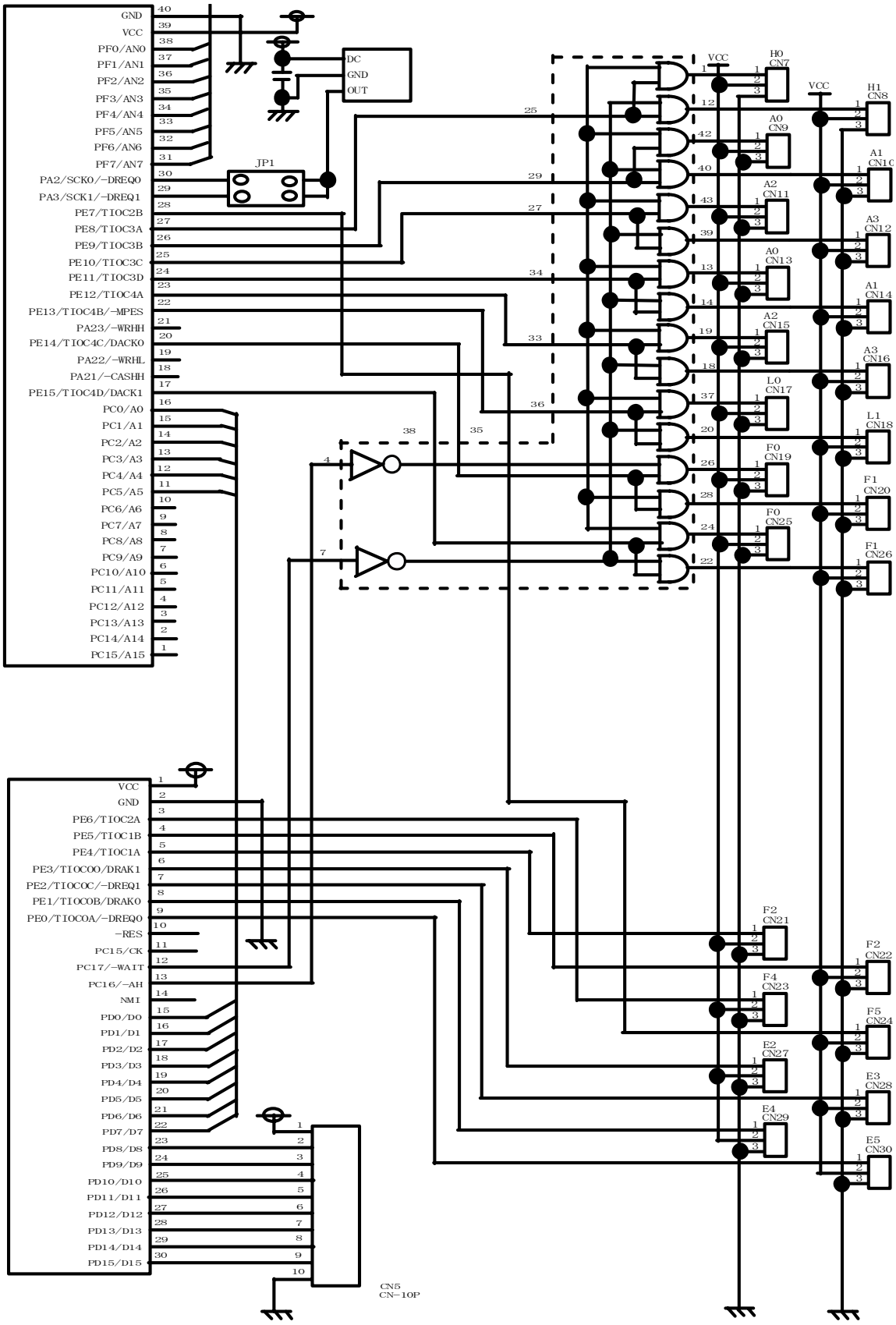

GND
VCC
PF0/AN0
PF1/AN1
PF2/AN2
PF3/AN3
PF4/AN4
PF5/AN5
PF6/AN6
PF7/AN7
PA2/SCK0/−DREQ0
PA3/SCK1/−DREQ1
PE7/TIOC2B
PE8/TIOC3A
PE9/TIOC3B
PE10/TIOC3C
PE11/TIOC3D
PE12/TIOC4A
PE13/TIOC4B/−MPES
PA23/−WRHH
PE14/TIOC4C/DACK0
PA22/−WRHL
PA21/−CASHH
PE15/TIOC4D/DACK1
PC0/A0
PC1/A1
PC2/A2
PC3/A3
PC4/A4
PC5/A5
PC6/A6
PC7/A7
PC8/A8
PC9/A9
PC10/A10
PC11/A11
PC12/A12
PC13/A13
PC14/A14
PC15/A15
DC
GND
OUT
JP1
VCC
H0 CN7
H1 CN8
A0 CN9
A1 CN10
A2 CN11
A3 CN12
A0 CN13
A1 CN14
A2 CN15
A3 CN16
L0 CN17
L1 CN18
F0 CN19
F1 CN20
F0 CN25
F1 CN26
F2 CN21
F2 CN22
F4 CN23
F5 CN24
E2 CN27
E3 CN28
E4 CN29
E5 CN30
VCC
GND
PE6/TIOC2A
PE5/TIOC1B
PE4/TIOC1A
PE3/TIOC0D/DRAK1
PE2/TIOC0C/−DREQ1
PE1/TIOC0B/DRAK0
PE0/TIOC0A/−DREQ0
−RES
PC15/CK
PC17/−WAIT
PC16/−AH
NMI
PD0/D0
PD1/D1
PD2/D2
PD3/D3
PD4/D4
PD5/D5
PD6/D6
PD7/D7
PD8/D8
PD9/D9
PD10/D10
PD11/D11
PD12/D12
PD13/D13
PD14/D14
PD15/D15
CN5
CN−10P

Fig.3-10 出力端子割り当て

3－4 バッテリー

本研究ではサーボモータ用と制御用の二つのバッテリーを使用している.サーボモータ用のバッテリーの仕様を表 3-3 に制御用のバッテリーを表 3-4 に示す.

表 3-4　サーボモータ用バッテリーの仕様

種類	ニッカド電池
定格電圧	7.2V
定格電流	600mA
重量	125g

表 3-4　制御用バッテリーの仕様

種類	ニッカド電池
定格電圧	8.4V
定格電流	150mA
重量	45g

４章　制御方法

４－１　制御方法

今回,本論文第 3 章　3.サーボモータ.で説明したサーボモータの制御方法は PWM 制御【パルス幅変調（Pulse Width Modulation）】とよばれる手法である.

４－２　PWM 制御

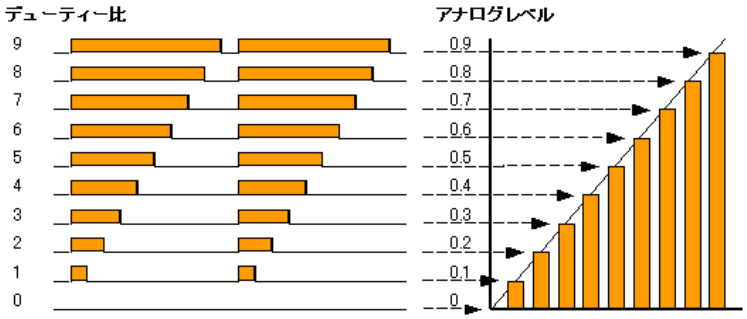


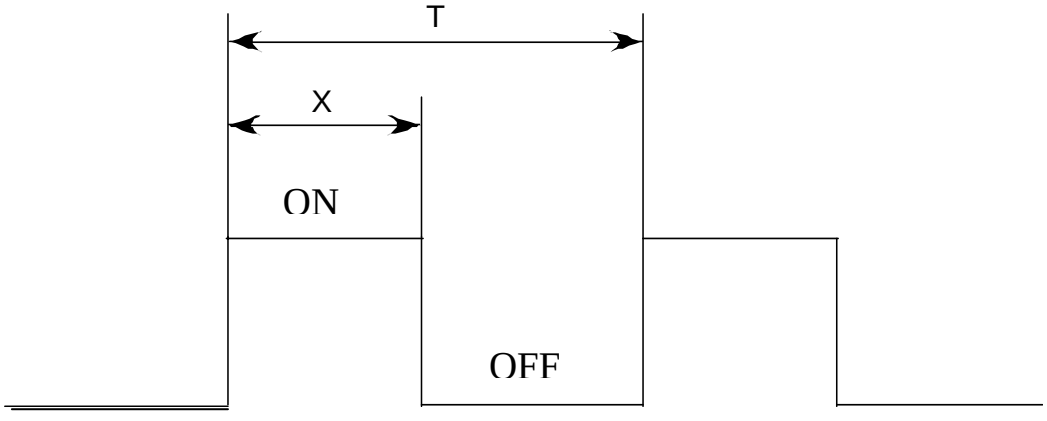
Fig4-1 PWM 制御

図の左側では，デューティー比が高いほどＯＮになっている期間が長く，デューティー比がゼロではずっとＯＦＦのままである様子を示しています.

モーターにこのようなタイミングで電源を与えてあげると，あたかもアナログ的に電源電圧を可変してあげたかのように，角度の制御が可能になることはこれが PWM による制御である.

４－３　デューティー比

PWM 制御を用いるにあたって，デューティー比が必要となってくる．ディーティー比は周期 T と，on 時間 x の比である.
これより以下のようになる.



$$duty = \frac{X}{T} \times 100[\%]$$

４－４　デューティー比とサーボモータの移動角度

４－４　デューティー比とサーボモータの移動角度

本研究では，デューティー比とサーボモータの移動角度の関係を調べるため，デューティー比１毎に移動角度をデータにとりさらに，それをパルス数に変換した．その表を，表 4-1 に示す．

表 4-1．デューティー比とサーボモータとパルス数の移動角度の関係

時間 Td [msec]	パルス数		角度(°)	デューティー比[%]
0.76	1260	1270	−75	15.2
0.8	1340	1190	−70	16
0.86	1420	1110	−65	17
0.9	1510	1020	−60	18
0.94	1590	940	−55	19
1	1670	860	−50	20
1.04	1760	770	−45	21
1.1	1850	680	−40	22
1.14	1930	600	−35	23
1.2	2020	510	−30	24
1.24	2100	430	−25	25
1.3	2190	340	−20	26
1.34	2270	260	−15	27
1.38	2360	170	−10	28
1.44	2440	90	−5	29
1.48	2530	0	0	29.7
1.54	2620	−90	5	30
1.58	2700	−170	10	31
1.64	2790	−260	15	32
1.68	2880	−350	20	33
1.74	2960	−430	25	34
1.78	3040	−510	30	35
1.82	3130	−600	35	36
188	3220	−690	40	37
1.92	3300	−770	45	38
1.98	3390	−860	50	39
2.02	3460	−930	55	40
2.06	3550	−1020	60	41
2.12	3630	−1100	65	42
2.16	3710	−1180	70	43
2.2	3780	−1250	75	44

横軸にディーティー比，縦軸にサーボモータの移動角度をとったグラフに Fig4-3 に表した. ただしこのデータは無負荷状態における関係でありまたデューティ比が 29.7%の時をニュートラル値として表している．ここで T＝8msec である．

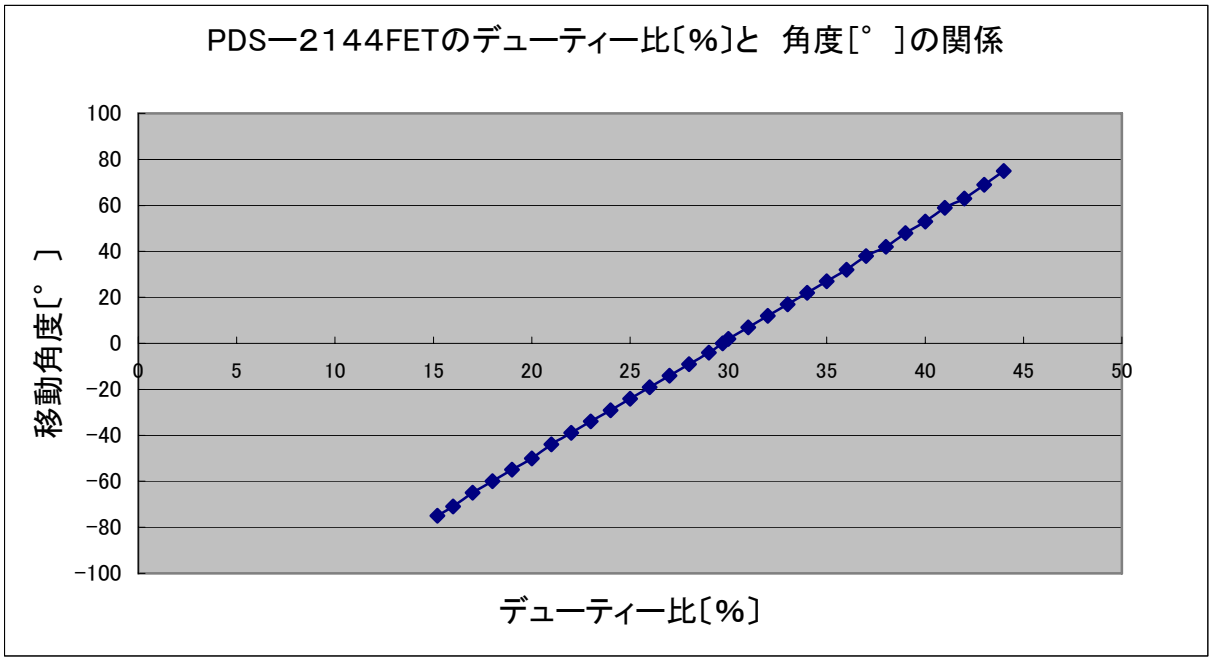


Fig4-3 デューティー比とサーボモータの移動角度の関係

これよりデューティー比とサーボモータの移動角度の関係は比例関係になっているとする。

４－５　パルス数とサーボモータの移動角度

本論文 4-4 節でデューティー比とサーボモータの移動角度の関係を説明したが，デューティー比では CPU が読み取ることができないため，CPU が読み取れる値に変えることが必要となってくる．

その読み取れる値パルス数と，デューティー比の関係は Fig4-4 のようになる．ただしこの状態は無負荷状態である．

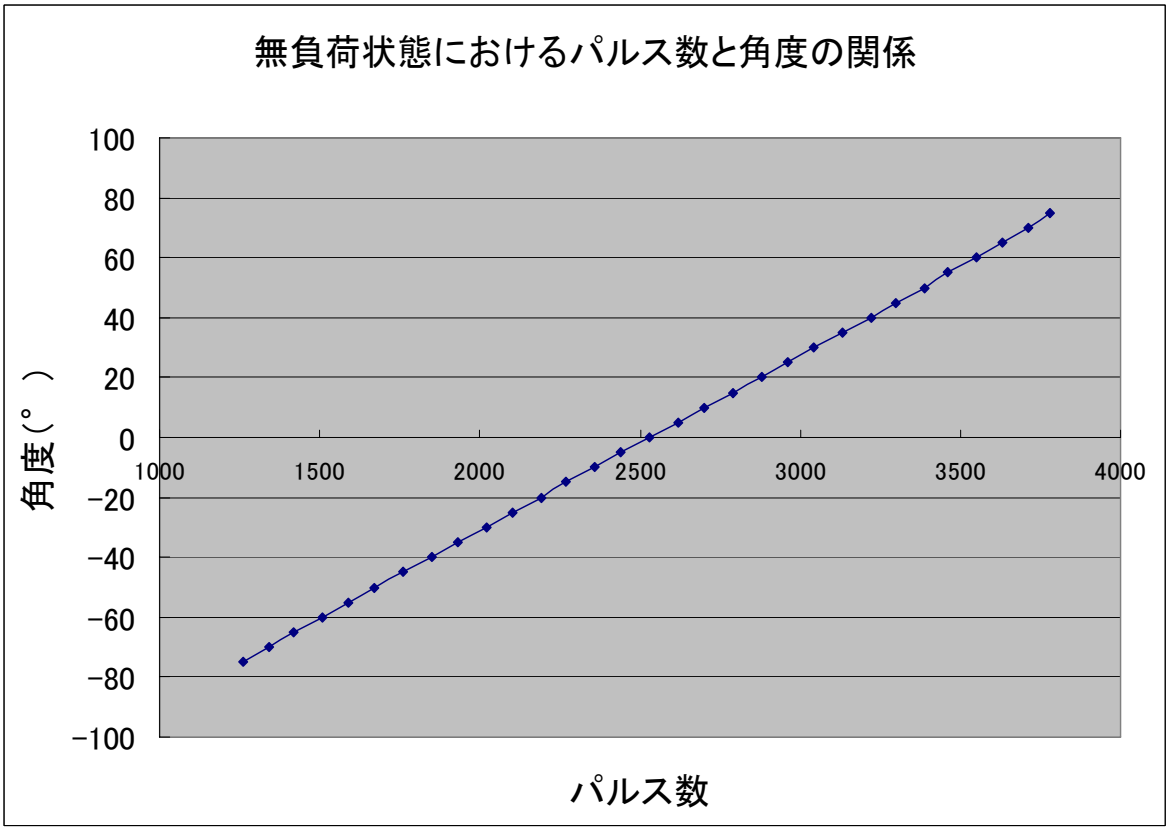


Fig4-4　パルス数とサーボモータの移動角度の関係

図中の直線は，

$$y = \frac{1}{16}(x - 2530) \qquad \cdots (4\text{-}1)$$

の関係にある．

ただしここでは

ｙ：角度

x：パルス数

である．また，2530 はニュートラルの値である．

４－６　割り込み処理プログラム

```
WDT．WRITE．TCSR＝0x5a26 ――――――――――④
a＝WDT．READ．TCSR．BYTE ――――――――――⑤
WDT．WRITE．TCSR＝0xa53d ――――――――――⑥
```

プログラム④では，使用したい時間を指定している．なお今回のプログラムでは，割り込みを，８ｍｓや５ｍｓで発生させたいために，プログラム①で最大時の時間を９．４ｍｓとしている

・５ａはアドレスの指定をしている．

・最後の２６は８ｍｓを表している．

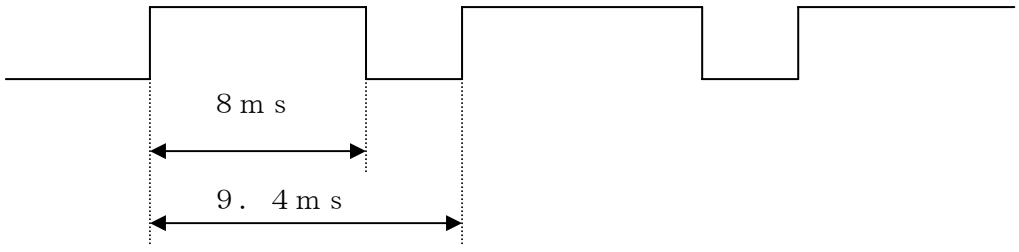
８ｍｓの算出方法

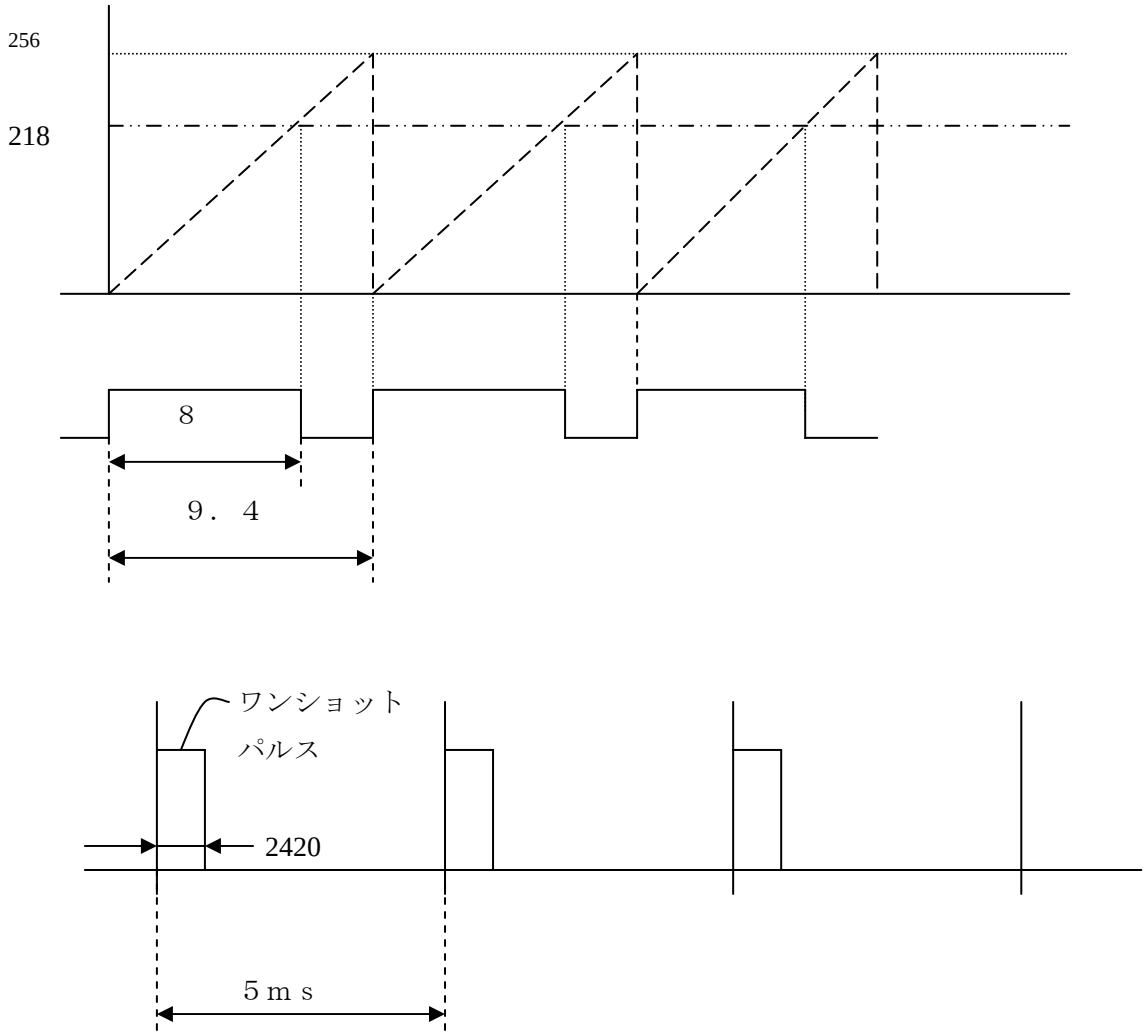
２６は１６進数なので（０ｘより），１０進数に変換すると，３８になる

※２６（１６進数）⟶ ００１０　０１１０（２進数)⟶ ３８（１０進数）

$$9.4 \times \left(\frac{256-X}{256}\right) = \text{使用したい時間}$$

Ｘに３８を入れると今回のプログラムで使っている値が出てくる.つまり５ｍｓを使用したい場合は，上式の使用したい時間に５ｍｓを代入し Ｘ を求め，その値をプログラムに入力する（ただし Ｘ は１０進数でてくるため．１６を使用する場合は，変換をしなければいけない.）





プログラム⑤は VOF クリアする
プログラム⑥でプログラムの左辺はプログラム④と全く同じである．プログラム⑥ TCSR への書き込みであるが関数が同じである．そのため右辺の最初でａ５とあるこれで TCSR への書き込みをしている．

４－７　　ウォッチドックタイマ

ウォッチドックタイマは本来は,システムの監視を行うためのタイマであるが,それと同時にインターバルタイマの機能があるのでこれを使用することができる．

インターバルタイマは,TCNT がオーバーフローするごとに,インターバル割り込みを発生する．この割り込みが発生したとき,１６本のタイマをワンショットで起動すれば,PWM を発生させることができる．

レジスタには**タイマコントロール／ステータスレジスタ（TCSR）,タイマカウンタ(TCNT),リセットコントロール／ステータスレジスタ(RSTCSR)**がありこれらを設定する必要がある.

1) **タイマカウンタ(TCNT)**

TCNT は,８ビットの読み書き可能なアップカウンタである．TCNT の値がオーバフロー（H'FF から H'００）すると,TCSR の OVF フラグが１にセットされる。ここで割り込みが発生するので,オーバーフローフラグを読み取り, TCNT に５msec のカウント値を書き込んでやる.

2) **タイマコントロール／ステータスレジスタ（TCSR）**

TCNT は,８ビットの読み書き可能なレジスタで,TCNT に入力するクロックやモードの選択を行う.

タイマモードセレクトは０でインターバルタイマである.

タイマイネーブルは１でカウントをスタートする.

クロックセレクトの CKS2－CKS０ビットで選択された内部クロックにより,カウントアップを開始する.

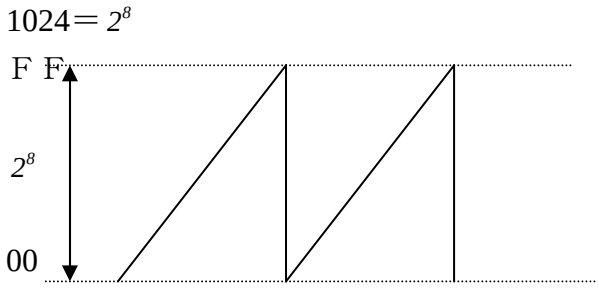
CKS2	CKS1	CKS0	
0	0	0	φ/2
		1	φ/64
	1	0	φ/128
		1	φ/256
1	0	0	φ/512
		1	φ/1024
	1	0	φ/4096
		1	φ/8192

OVF	WT/T	TME	---	---	CKS2	CKS1	CKS0

例えば，ここに 00111101Ｂ（＝3d（１６進数））を入れてやると

0	0	1	1	1	1	0	1

となる．これに対応しているＣＫＳをみると，φ/1024 となる.



$$\frac{28MH_Z}{1024} = 27343.75\,H_Z$$

$$T \quad = \frac{1}{27343.75}\ \mathrm{S} = 0.03657 \quad \mathrm{ms}$$

よってフルカウントでの時間インターバル

$$T_f = 0.03657 \quad \mathrm{ms} \quad \times \quad \frac{256}{28} \quad = \quad 9.36 \quad \mathrm{ms} \quad \fallingdotseq \quad 9.4 \quad \mathrm{ms}$$

※ここで, フルカウント値は計算で　9.36　ms　と算出された, しかしロボットマガジンNo21. P49 を見るとフルカウント値に 9.4 があるため, ここでは 9.4　ms としている.

例

WDT.WRITE.TCSR = 0x5a26

このときオシロスコープを使用し，周期を計測したところ,

周期＝　125　Hz　＝　８msec

となった．この結果を計算により，算出してみる

５a はアドレスの指定のためここでは, 下位 2 ビットの 26 を使用する.

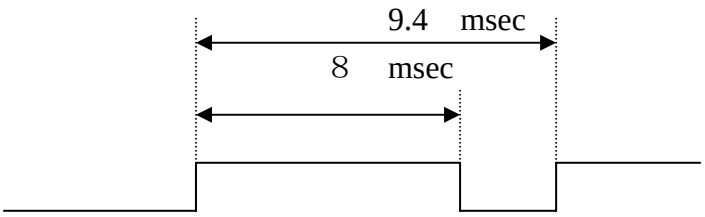
まず，26 を 16 進数に変換する.

26＝0010　0110　（H）

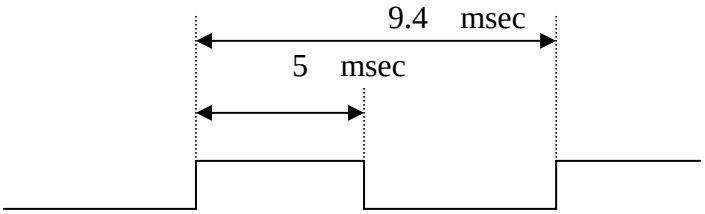
となる．これを，10 進数に変化する.

0010　0110　＝　26

$$9.4 \quad \mathrm{msec} \quad \times \quad \left(\frac{256-28}{256}\right) \quad = \quad 8 \quad \mathrm{ms}$$



周期を 5msec にするためには



$$9.4 \text{ msec} \times \left(\frac{256 - X}{256}\right) = 5\text{msec}$$

X ＝ 120

この値を，二進数に変換する。0111 1000　となる.

この二進数を１６進数に変換すると，7 8

WDT.WRITE.TCSR = 0x5a78

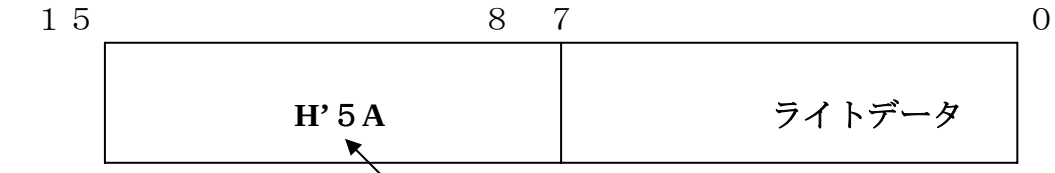
とすれば，周期　5ms となる.

3) **TCSR, TCNT への書き込み**

TCSR, TCNT ともに同一アドレスになっているため,TCNT, TCSR へ書き込むときには,下位バイトを書き込みデータに,**上位バイトを H’５A（TCNT のとき）**または **H’A5（TCSR のとき）**にしてワード転送を行う．プログラム例をプログラム１に示す.

＜TCNT ライト時＞

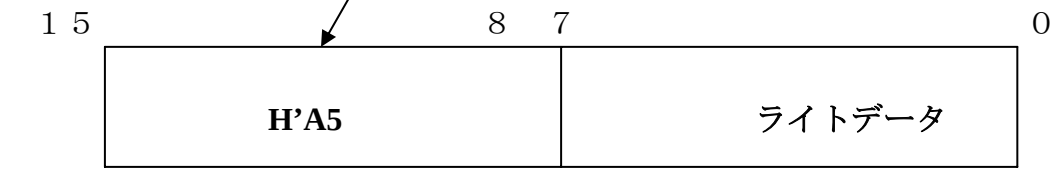
アドレス　　H’FFA8



上位に書き込むのは決まっている.

＜TCSR ライト時＞

アドレス　　H’FFA8



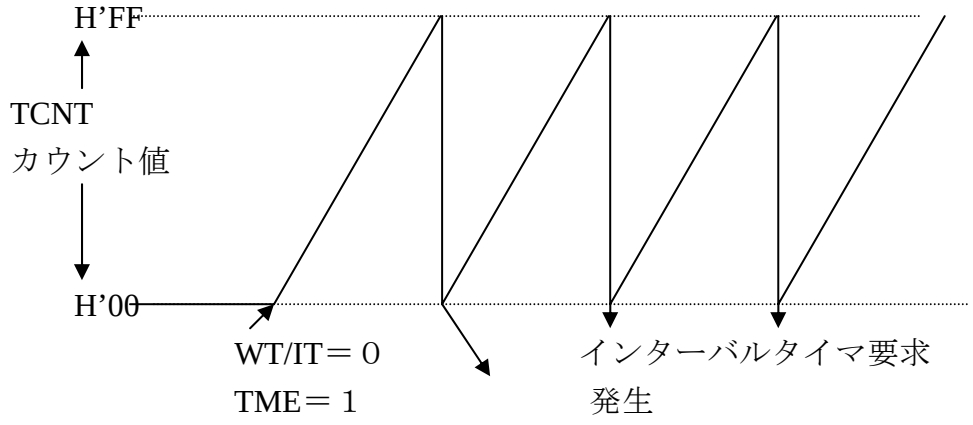
4)インターバルタイマ時の動作

インターバルタイマとして使用するには,**TCSR** の **WT/IT** ビットを０にクリアし,**TME** ビットを１にセットする.

TCNT がオーバフローするごとに,インターバルタイマ割り込み要求が発生する．これにより,一定時間ごとにインターバルタイマ割り込みを発生させることができる.

プログラム１

```
Char a;
WDT.WRITE.TCSR=0x5a26; //TCNT=117=0x75 5msec     5a で TCNT に書き込み
A=WDT.READ.TCSR>BYTE;//OVF クリア
WDT.WRITE.TCSR=0xa53d;//a5 で TCSR への書き込み
```



ＷＤＴの設定プログラム

```
ＷＤＴ．ＷＲＩＴＥ．ＴＣＳＲ＝０ｘａ５３ｄ ――――――①
ＩＮＣ．ＩＰＲＨ．ＷＯＲＤ ＝０ｘｆ０００ ――――――②
ＳｅｔＳＲＲｅｇ （０） ――――――③
```

①のプログラムでは，ＴＣＳＲ（タイマコントロール/ステータスレジスタ）の指定をしている．ＴＣＳＲは読み出し/書き込み可能な８ビットのレジスタで，タイマカウンタ（TCNT）に入力するクロック，モードの選択などを行います.

7	6	5	4	3	2	1	0
OVF	WT/IT	ＴＭＥ	――	――	CKS2	CKS1	CKS0
0	0	1	1	1	1	0	1
R/(W)	R/W	R/W	R	R	R/W	R/W	R/W

ビット７：オーバフローフラグ（OVF）

ビット７	説明
OVF	
0	インターバルタイマモードで TCNT のオーバフローなし （初期値） ［クリア条件］ OVF を読み出してから０を書き込む
1	インターバルタイマモードで TCNT のオーバフロー発生

ビット７～０には，①のプログラムの ３ｄ が対応している。よってここでは０が入力されている.

ビット６：タイマーモードセレクト（WT/IT）

ウォッチドックタイマとして使用するか，インターバルタイマとして使用するかを選択します．この選択によって，TCNT がオーバフラウしたとき，インターバルタイマ割り込み（ITI）が発生するか，WDTOVF 信号が発生するかが決まります.

ビット６	説明
WT/IT	
0	インターバルタイマモード:TCNT がオーバフラウしたとき CPU へインターバルタイマ割り込みを要求 （初期値）
1	ウォッチドックタイマモード：TCNT がオーバフラウしたとき WDTOVF 信号を外部へ出力

今回のプログラムでは，０を選択している．

ビット５：タイマイネーブル（TME）

タイマ動作の開始または停止を設定します．

ビット５	説明
TME	
０	タイマディスエーブル：TCNT を H’００に初期化し，カウントアップを停止（初期値）
１	タイマイネーブル：TCNT はカウントアップを開始。TCNT がオーバフローすると WDTOVF 信号または割り込みを発生

今回のプログラムでは１を選択している．

ビット４，３：予約ビット

読み出すと常に１が読み出されます。書き込む値もつねに１にすること．

ビット２～０：クロックセレクト２～０（CKS２～CKS０）

システム（φ）を分周して得られる８種類の内側クロックから，TCNT に入力するクロックを選択します．

ビット２	ビット１	ビット０	説明	
CKS２	CKS１	CKS０	クロック	オーバーフロウ周期（φ＝28MHz の場合）
０	０	０	φ/2	0.018ｍｓ
		１	φ/64	0.585ｍｓ
	１	０	φ/128	1.2ｍｓ
		１	φ/256	2.3ｍｓ
１	０	０	φ/512	4.7ｍｓ
		１	φ/1024	9.4ｍｓ
	１	０	φ/4096	37.4ｍｓ
		１	φ/8192	74.9ｍｓ

今回のプログラムでは,１０１を選択

②のプログラムでは優先レベルのしていを行っている。割り込み優先レベル設定レジスタ A～H（ＩＰRA～ＩＰRH）は，それぞれ読み出し/書き込み可能な１６ビットのレジスタで構成されている．今回のプログラムでは，ＩＰRH を使用している．これはＷＤＴを使用するためである．

レジスタ	ビット			
	１５～１２	１１～８	７～４	３～０
割り込み優先レベル設定レジスタ A	IRQ０	IRQ１	IRQ２	IRQ３
割り込み優先レベル設定レジスタ B	IRQ4	IRQ5	IRQ6	IRQ7
割り込み優先レベル設定レジスタ C	DMAC０	DMAC1	DMAC2	DMAC3
割り込み優先レベル設定レジスタ D	MTU0	MTU0	MTU1	MTU1
割り込み優先レベル設定レジスタ E	MTU2	MTU2	MTU3	MTU3
割り込み優先レベル設定レジスタ F	MTU4	MTU4	SCI0	SCI1
割り込み優先レベル設定レジスタ G	A/D	ＤＴＣ	ＣＭＴ0	ＣＭＴ1
割り込み優先レベル設定レジスタ H	WDT，BSC	I/O	予約	予約

③のプログラムは割り込みマスククリアをしている

また，PWM の切り替えず Fig．4-5 に示す.

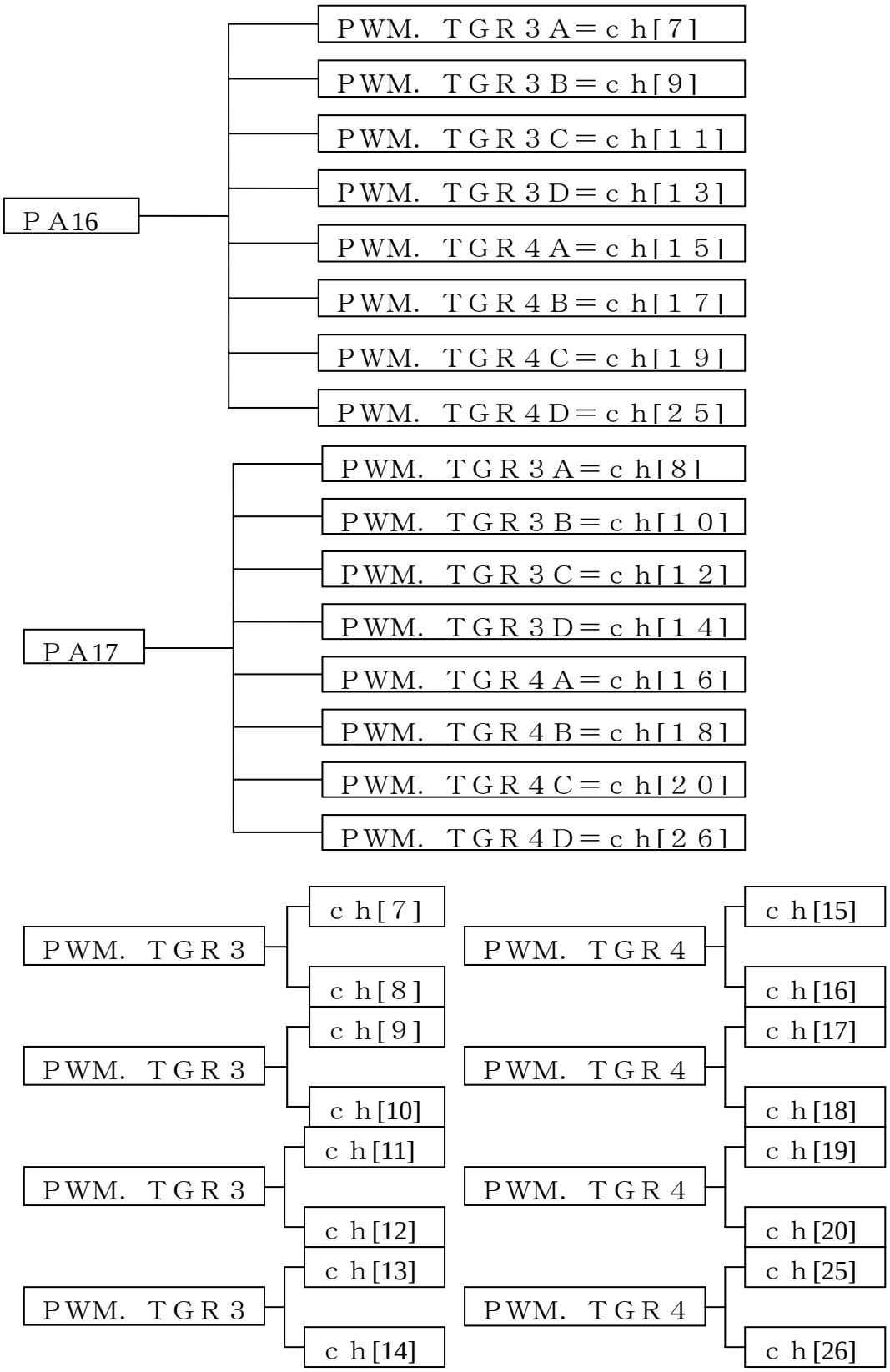
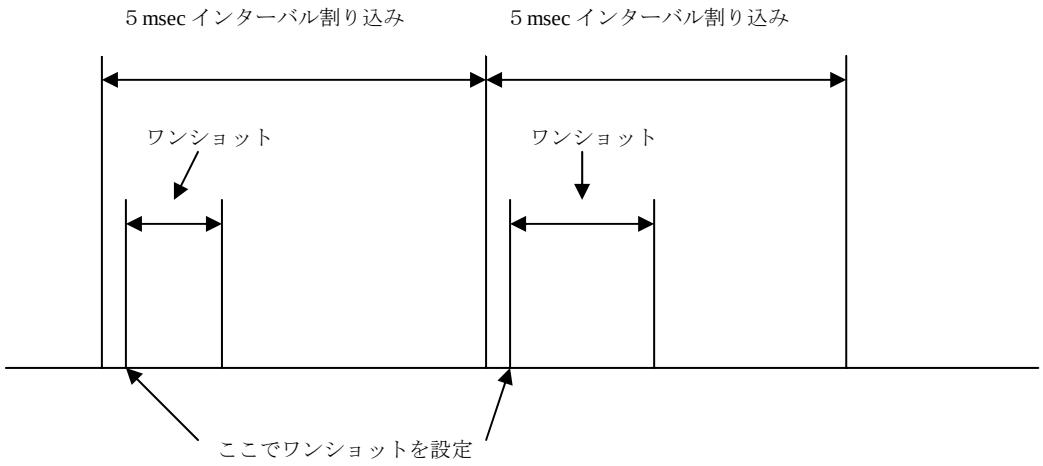


Fig 4-5 PWM の切り替え図

４－8　　１６軸の PWM を出力する

うまく組み合わせても出力可能な最大の PWM 出力数は１２となる. そこで図１に示すように,WDT のインターバルタイマを使用して周期を決定する.



割り込みと同時にワンショットタイマを起動し,デューディーを決定する．こうすることによって１６ｃｈすべてのポートから PWM を出力することができる．つまり,普通は一つの信号で一つのモータを制御するところを,一つの信号をプログラムによって分割し一つの信号で複数のモーターを制御しようというもの.

4-9　　MTU について

MTU0．TCR．BYTE　＝　０ｘ０２

TCR：タイマコントロールレジスタ　（８ビットレジスタ）

タイマコントロールレジスタ（TCR）は各チャンネルの TCNT カウンタを制御するレジスタです．MTU には，チャンネル０～４に各一本，計５本の TCR レジスタがあります．TCR レジスタは，８ビットの読み出し/書き込み可能なレジスタです．パワーオンリセットまたはスタンバイモードで H’00 に初期化されます．マニュアルリセットでは初期化されません.

CCLR2	CCLR1	CCLR0	CKEG1	CKEG0	TPSC2	TPSC2	TPSC2
0	0	0	0	0	0	1	0

CCLR2～0：TCNT のクリアの禁止
CKEG1～0：立ち上がりエッジ
TPSC2～0：内部クロック　φ/16 でカウント
φ/16＝28［MHz］/16≒0．57μsec

MTU0．TMDR．BYTE　＝　０ｘＣ０

TMDR：タイマモードレジスタ，８ビットレジスタ
タイマモードレジスタ（TMDR）は各チャンネルの動作モードの設定を行います．MTU には，各チャンネル１本，計５本のレジスタがあります．TMDR レジスタは，８ビットの読み出し/書き込み可能なレジスタです．パワーオンリセットまたはスタンバイモードで H’C0 に初期化されます. マニュアルリセットでは初期化されません.

—	—	BFB	BFA	MD3	MD2	MD1	MD0
1	1	0	0	0	0	1	0

MTU0．TIOR．WORD　＝　０ｘ５５５５
MTU0．TIOR．BYTE．H＝０ｘ５５
MTU0．TIOR．BYTE．L＝０ｘ５５

TIOR：タイマ　I/O　コントロールレジスタ
タイマ　I/O　コントロールレジスタ（TIOR）は TGR を制御するレジスタです．チャンネル０，３，４に各二本，チャンネル１，２に各一本，計８本の TIOR レジスタがあります．TIOR レジスタはパワーオンリセットまたはスタンバイモードで H’00 に初期化されます．マニュアルリセットでは初期化されません.

MTU0．TIOR．BYTE．H＝０ｘ５５　に関して

IOB3	IOB2	IOB1	IOB0	IOA3	IOA2	IOA1	IOA0
0	1	0	1	0	1	0	1

IOB3～0：TGROB に関する設定　　初期値１でコンペアマッチで０を出力
IOA3～0：TGROA に関する設定　　初期値１でコンペアマッチで０を出力

MTU0．TIOR．BYTE．L＝０ｘ５５　に関して

IOD3	IOD2	IOD1	IOD0	IOC3	IOC2	IOC1	IOC0
0	1	0	1	0	1	0	1

IOD3～0：TGROD に関する設定　　初期値１でコンペアマッチで０を出力
IOC3～0：TGROC に関する設定　　初期値１でコンペアマッチで０を出力

チャンネル　MTU０，３，４　は　２チャンネルの出力
チャンネル　MTU１，２　は　１チャンネルの出力　　｝モード１の場合

MTU0，TCNT　＝　０　について
TNCT：タイマカウンタ　１６ビット

タイマ TNCN カウンタ（TCNT）は１６ビットのカウンタです．各チャンネルに一本，計５本の TCNT カウンタがあります．TCNT カウンタはパワーオンリセットまたはスタンバイモードで H’00 に初期化されます．マニュアルリセットでは初期化されません. TCNT カウンタの８ビット単位でのアクセスは禁止です．常に１６ビット単位でアクセスしなくてはいけません.

MTU．TSTR．BIT．CST0＝１
TSTR：タイマスタートレジスタ　８ビット

CST4	CST３	—	—	CST2	CST1	CST0
0	0			0	0	1

チャンネル０→４本
チャンネル１→２本
チャンネル２→２本
チャンネル３→４本
チャンネル４→４本
} 計１６本をコントロール

4-10 時間の作り方：１／１７

ロボコンマガジン No. ２４　　プログラム参照

```
  //時間のカウント
tim5ms++;tx5++;          //8ms のカウンタインクリメント（８msec 毎に増える）
                         //125 回になったら 1 秒単位のカウンタインクリメント
if(tim5ms>125){
  tim1sec++;             // tim1sec は一秒毎に 1 増える
  tim5ms=0;
}
else{
                         //歩行用カウンタ txt デカウントアップ
if(tx5>txt){
  txx++; tx5=0;
}
else{
}
```

tim1sec　，txx は初期化するまで増え続ける

・ 動作時間の変更の仕方

時間を遅くするには，txt を大きくすればよい．

もし，txt＝５であったとすると．

```
if(tx5>txt){          //８msec のカウンタ　０．００８sec×５＝４０msec
  txx++; tx5=0;       // txx は４０msec ごとに増える
```

```
}
else{
}
```

<u>または，tim を増やしてやる．</u>

腰を落とすプログラム

```
void west_down(int ko, int tim){    // ko tim にそれぞれ 30    30 を入れる
        tx5=0;txx=0;
        while(txx<=2*tim){              // txx は８msec 毎に１増える．
                                          0≦txx≦60←分割数にも対応
                r_xz(0,ko*txx/(2*tim));
                l_xz(0,ko*txx/(2*tim));
        }
        //r_xz(0,ko);
        //l_xz(0,ko);
}
```

８×６０msec の時間がかかる

while 文の中はこの設定では８×60msec 掛かることになる．

よって

時間を早めるには，２通りある．

- txt を小さくする．
- ループの２×tim を小さくする．

第五章　設計の検討

5-1　機械部品

モータを用いて足を組み立てるためには，ブラケットが必要である．大半は市販品を使用しているが，市販品だけではすべてを組み立てることができないために，必要なパーツをアルミ材を加工し製作した．その際に膝と足首間のパーツにおいては軸間距離精度がかなり必要となった．これは，後に制御する際に股関節と膝関節間距離と膝関節と足首間距離が同じだと，制御が行いやすくなるためである．また足首には，ロール軸とピッチ軸を持たせるために 2 つのモータを使用している．ここでのパーツでは軸をしっかりとることが重要であり，ここで軸の精度を間違えると，重心移動した際に左右で異なる角度に曲がることになるため，モータに負荷を与えてしまうことになってしまう．このようなことより，パーツを作成する際には，精度を出すために注意を払った．Fig.5-1 に二足歩行の全体図を示す．

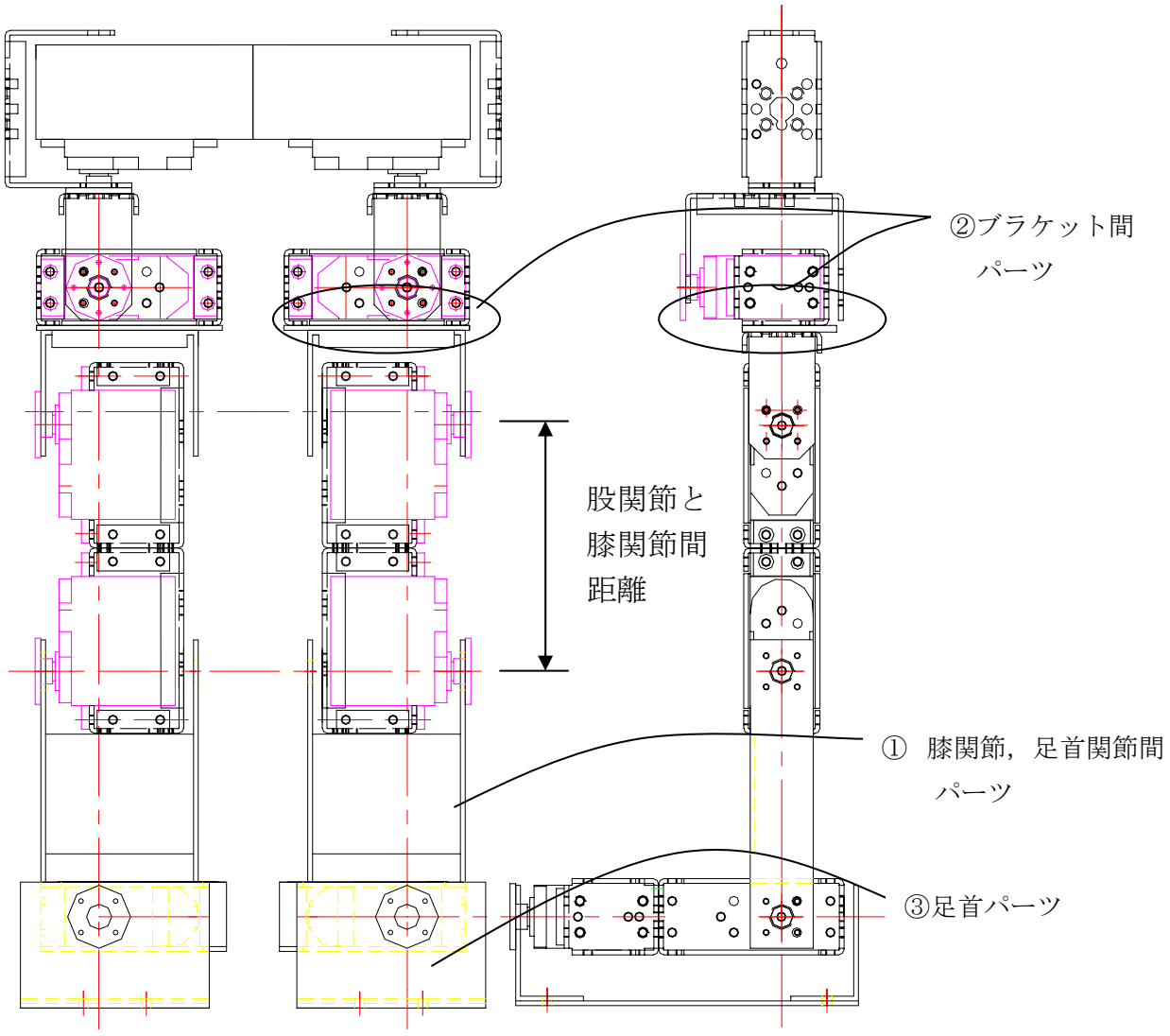


Fig.5-1　二足歩行の全体図

5-1-1　①膝関節，足首関節間パーツ

Fig.5-1-1 のパーツ①は，膝関節と足首を繋ぐパーツである．このパーツでもっとも重要とされるのが次の３項目である．

- 軸間距離
- 板材に軸穴が垂直である
- ２軸が平行である

軸間距離は，股関節と膝関節間距離と同じピッチにすることで，制御がしやすくなるという利点があるためである．ここでの制御がしやすいというのは，重心に主に関連している．板材に軸穴が垂直でなければならないのは，もしこの軸穴が左右でずれていたなら，組み立てた際に，モータとパーツが傾いた状態で組上がることになる．そうなると，いくら良い制御をしても，望んでいる動作をさせるには困難である．

２軸が平行でなければならないのは，これも組み上げたとき，そして制御する際に問題が生じる．軸穴が平行でなければ当然 2 つのモータの軸もずれることになる．これも先ほど述べたようにいくら良い制御をしても，望んでいる動作をさせるには困難である．ということになる．

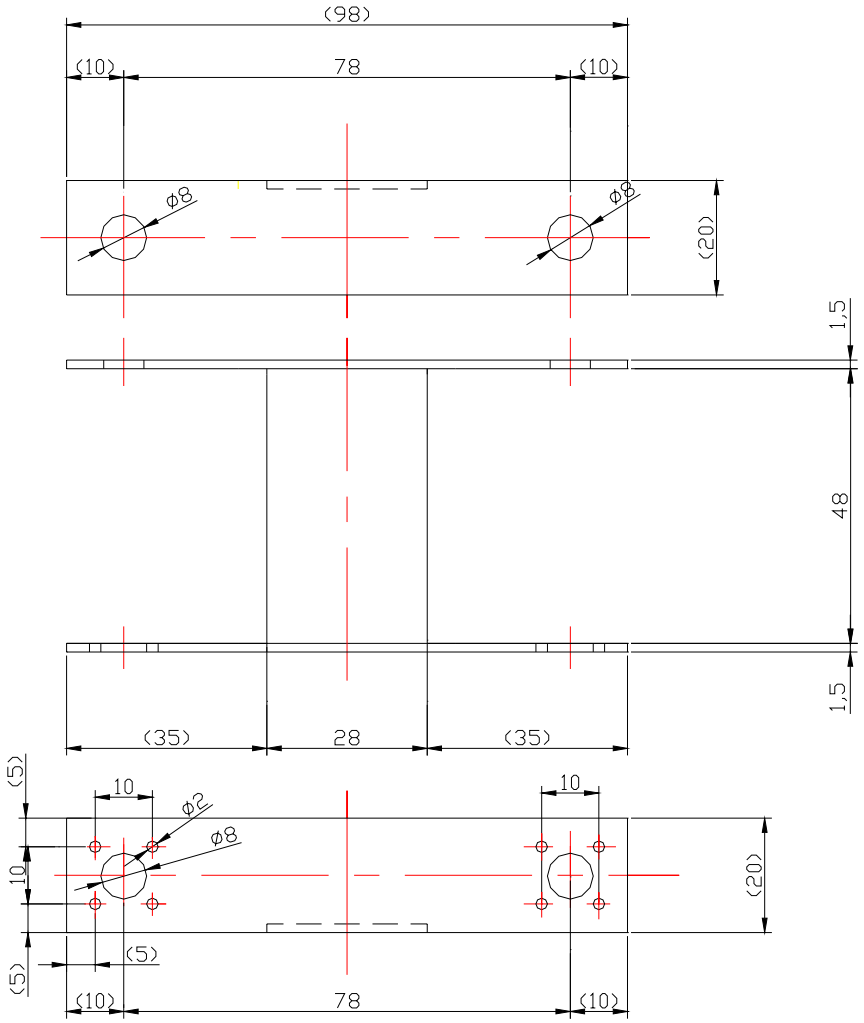


Fig.　5-1-1　①膝関節・足首関節間パーツ

5-1-2　②モータブラケット間接合部材

Fig.5-1-1 のパーツ②は ，足首関節（左右方向）の軸と，股関節（左右方向）の軸を同軸上にそろえるための，パーツである．Fig.5-1-2 に示されているブラケットを組み合わせただけでは足首関節（左右方向）の軸と，股関節（左右方向）の軸が，ずれてしまう．ずれた状態だと制御が困難になるために，この部材を製作した．Fig.5-1-2 にこの部材の図面を示す.

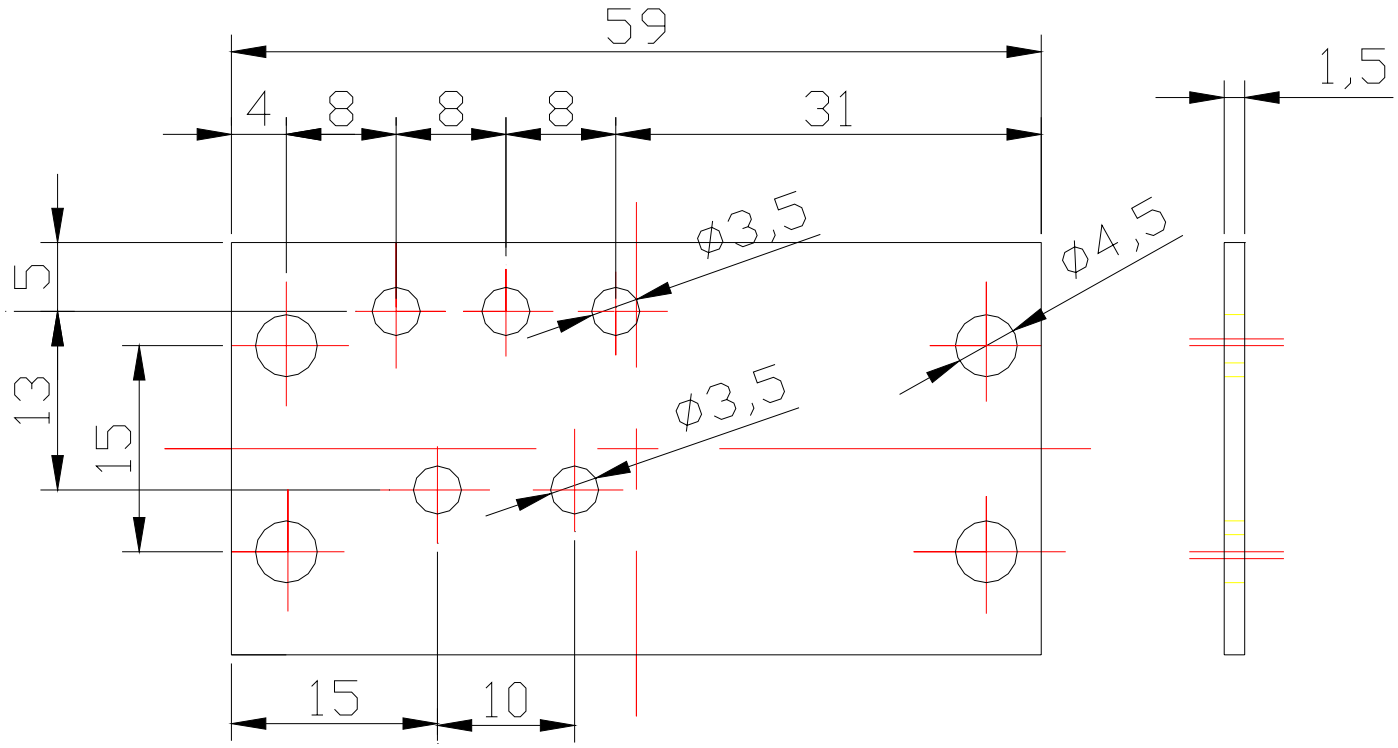


Fig.5-1-2　②ブラケット間パーツ

5-1-3　③足首パーツ

Fig.5-1-1 のパーツ③は，軸穴が重要となる．この軸穴は，重心を移動するためにある．つまりこの軸がずれると，正確な重心位置が得られなくなり，バランスを崩すことになる可能性がある．また，軸穴間距離も重要である．Fig.5-1-3 にこの部材の図面を示す.

5-2. 制御方法

良い制御をするためには，気をつけなければいけない点がある．それは，寸法制度の高い部品を用いることである．どんなによい制御アルゴリズムがあったとしても，機械部品が悪かったらパフォーマンスの限界は機械部品で決まってしまうし，コントローラの設計が悪ければせっかく設計したアルゴリズムを実装できなかったりする．

「良い制御」とは，制御をしようとしている対象を安定な状態にできることである．例えば，歩行が制御できているということは安定に歩行できていることになる．また「良い制御」をするためには，次のことが重要である．

- 即応性…入力に対して素早く反応できること
- ロバスト性…負荷や条件が変わっても安定でいられること
- 精度…出力が希望する値に高精度で一致できること

今回一番苦労した点はロバスト性である．歩行をするプログラムを作成し，空中につるし負荷をかけない状態で実行させた場合においては，ロボットは予測している動作を行うが，接地した状態において実行するとバランスを崩し転倒してしまった．この転倒する場合というのは片足をあげる際であった．つまり，ロボットの重量をｍとした場合に，それまで両足にかかっていた重量ｍが片足をあげることで，支持脚（足を上げた状態において，床に接している足）で支えなければいけないことになる．例えば，右足をあげた場合には，左足が支持脚となる．このとき左足の股関節左右方向のモータ，及び左足首の左右方向のモータには，反時計回りの力が働くことになる．このことを考慮にいれ，左足の股関節左右方向のモータ，及び左足首の左右方向のモータに，時計回りの動作をさせ．安定を保つようにしている．具体的な内容は付録 No.17 に示す．

5-3.プログラム

プログラムについては，C 言語を用いて作成した．プログラムは，屈伸プログラム，重心移動プログラム，足踏みプログラム，歩行プログラム（静歩行）を作成した．これらのプログラムで，足踏みプログラム，歩行プログラムについては，２通りの異なるプログラムを作成した．我々は，一方を座標間を補間する数値によるプログラムと呼び，もう一方を計算によるプログラムと呼んでいる．

数値によるプログラムは，スタート位置の角度データとエンド位置の角度データを 12 個のサーボモータにそれぞれ決めてやり，スタートデータからエンドデータまでを，何分割かにする．そのときこの分割数によりスピードが変わってくる．ただし，スタートデータとエンドデータの差は各モータによって異なるため，注意が必要である．また数値によるプログラムでは，分割数をエンドデータに近づくにつれ増やすようにプログラムをくんでやれば，初め速く動かしだんだん遅くなるようにすることが可能であると考えられる．つまりこれにより，慣性力を考慮していない制御ができる可能性があることがわかる．

数値によるプログラムに比べて，計算によるプログラムでは，位置の軌跡に対して正弦関数を用いることによって，より滑らかな歩行を可能にした．数値によるプログラムでは，直線的な動きになっていたが，計算によるプログラムでは曲線的な軌跡を描いた動作が可能となった．ただし，計算によるプログラムでは，実行する際に常に計算を行っている状態になっているので，どうしても，バイト数が大きくなるという欠点もある．

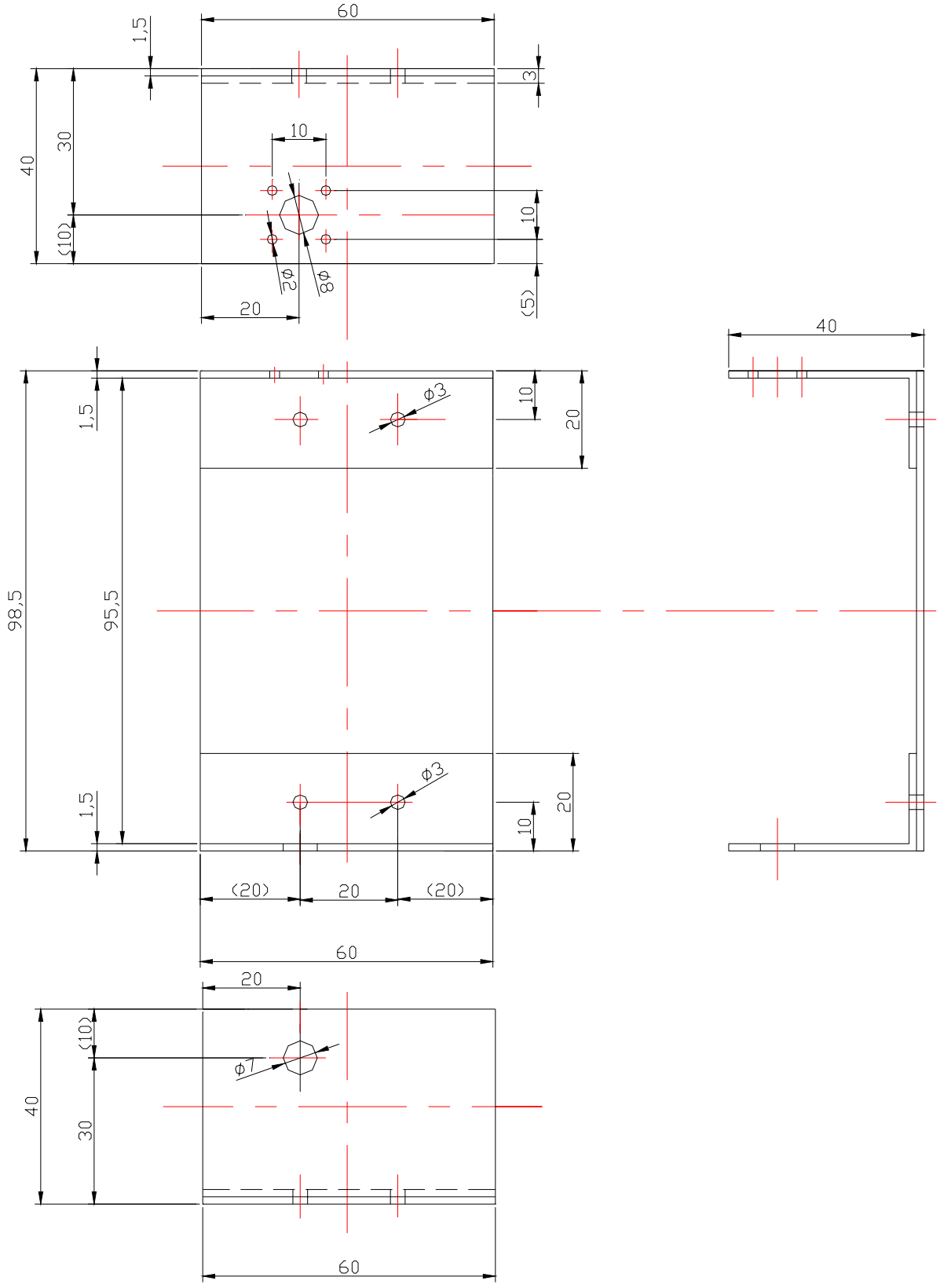


Fig. 5-1-3　　③足裏パーツ組図

六章　動作実験の検討

この章では，歩行動作の基本である，重心下げ，重心移動，屈伸，足踏みについての検討をし，各動作におけるサーボモータの制御方法について述べるそして，最後にこれらの基本動作を基にした，二足歩行の動作アルゴリズム及びサーボモータの動きについて解説する．

6-1　重心下げ

Fig.6-1-1 は腰を少し落とした状態である．例えばmr0 とは右側の 0 番関節のモータを示す。同様に、ml0 であれば、左側の 0 番関節モータを示す。この Fig.6-1-1 は我々が制作したロボットの簡略化した図である．

また重心下げにおいて、mr2 ml2 ml3 mr3 ml4 mr4 のモータを用いる。

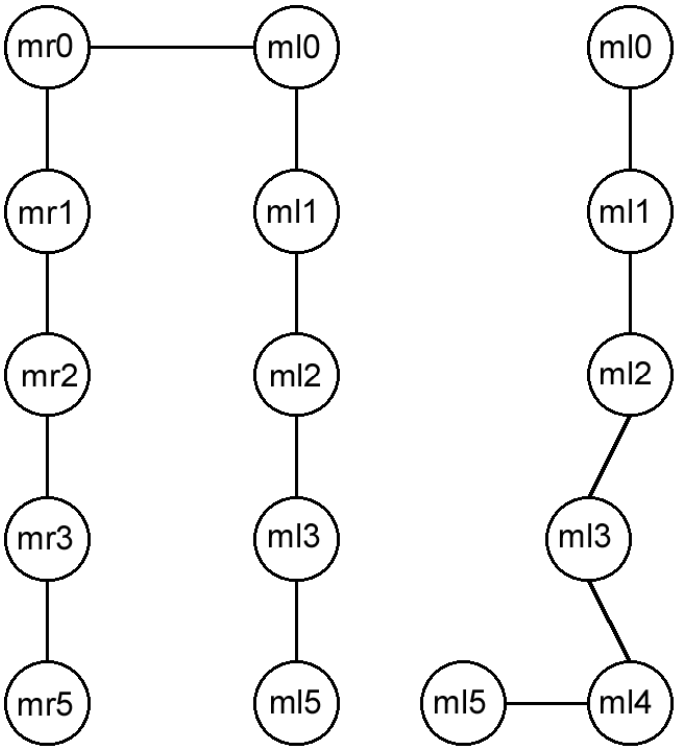


Fig.6-1-1 直立姿勢図

6-2　重心移動

この STEP1～2 では腰を指定した値だけ下げた状態から，指定した値だけ左右に重心移動する．ここでは左への重心移動を例にとり説明する．また，ここで腰を常に弱冠落とし気味なの，は重心をできるだけ低い位置に設定するためである。

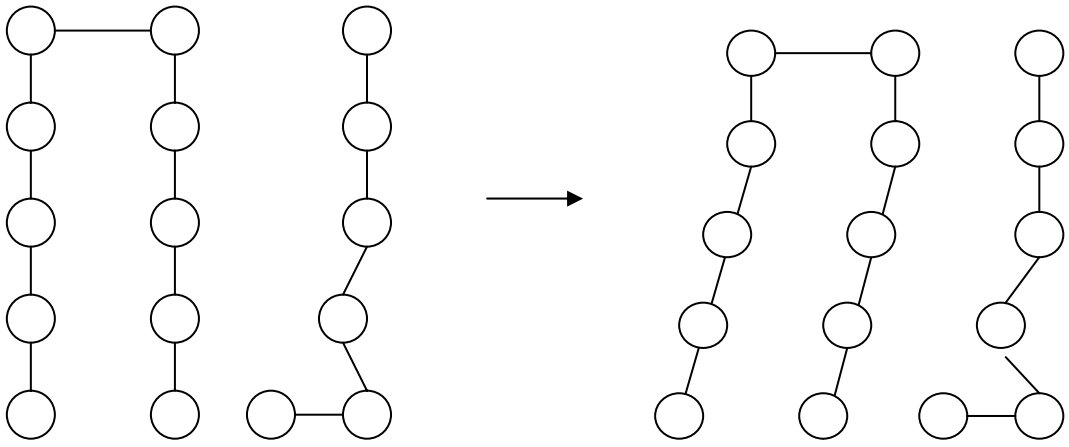


Fig.6-2-1　Step1

Fig.6-2-2　Step2

左に移動したい分の値をプログラム上で入力する．（ただし，単位［mm］）
入力された値は，プログラム上で計算され，各モータの回転角度を算出する（ここではｍｒ１，ｍｒ５，ｍｌ１，ｍｌ５を駆動させる）．その算出された値はさらに，プログラム上においてパルス数に変換され，各モータに送られ，実行されるようになっている．具体的な算出方法は、付録に示す。また重心移動のフローチャートを Fig.6-2-4 に示す.

例．左への重心移動

Fig.6-1-4 に示すように移動したい分の値を足の長さで割り，その算出結果を三角関数を用いて角度を算出し，その角度をパルス数に変換している．

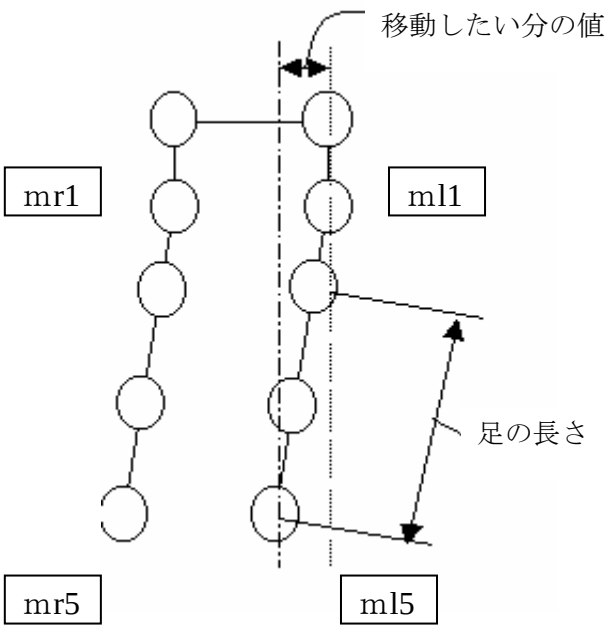


Fig.6-2-3 左への重心移動

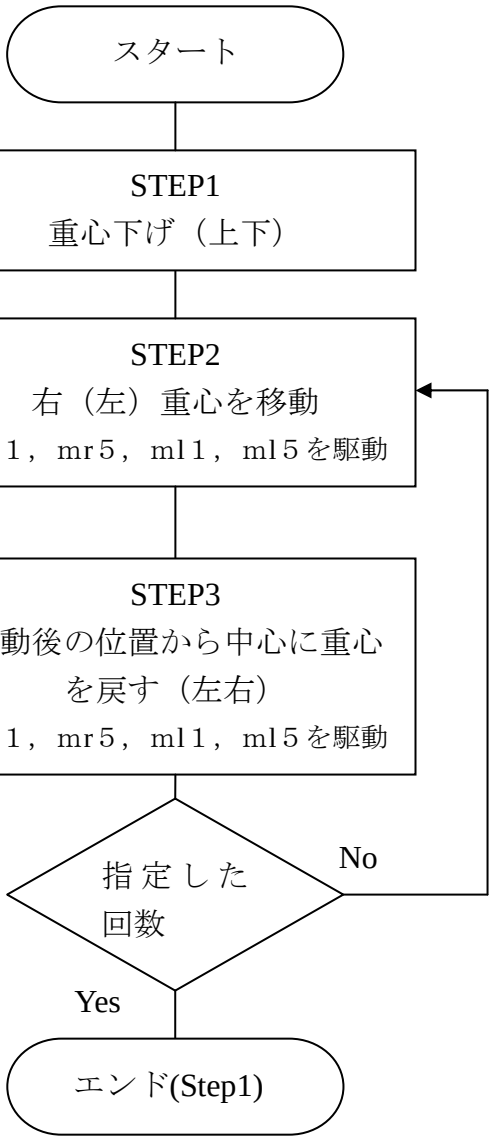


Fig.6-2-4　重心移動フローチャート

6-3　屈伸

腰を指定した値だけ下げた状態から，指定した値だけさらに腰を下げる．

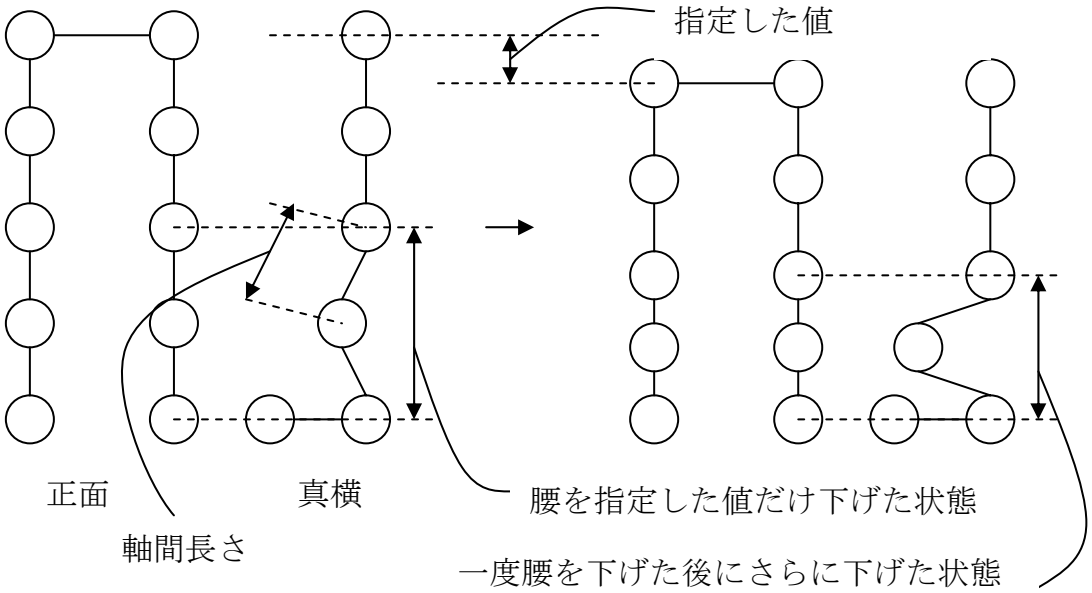


Fig.6-3-1　Step1　　Fig.6-3-2　Step2

この Fig.6-3-1 と Fig.6-3-2 を繰り返すことによって屈伸になる. この移動角度は, Fig.6-3-2 の一度腰を下げた後にさらに下げた状態を二倍した軸間距離で割り，その算出結果を三角関数で角度を算出する．ここでの駆動モータはmr2，mr3，mr4，ml2，ml3，ml4 となっている．屈伸のフローチャートを Fig.6-3-3 に示す

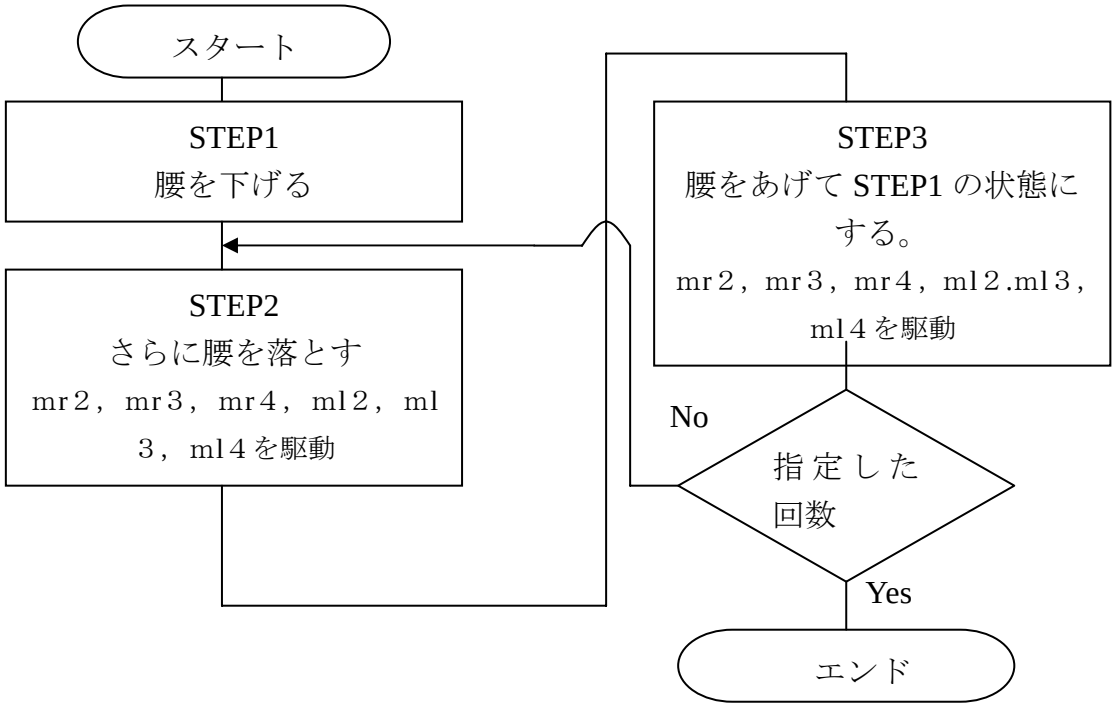


Fig.6-3-3　屈伸フローチャート

6-4 足踏み

重心移動をした後に，片足をあげる．つまり重心移動した後に片足だけ，屈伸させてやればよい．しかし片足をあげる際に，支持脚にはトルクが掛かるため，これを考慮してやる必要がある．ここでの駆動モータはmr1，mr2，mr3，mr4，mr5，ml1，ml2，ml3，ml4，ml5 となっている．足踏みのフローチャートを Fig.6-4-4 に示す.

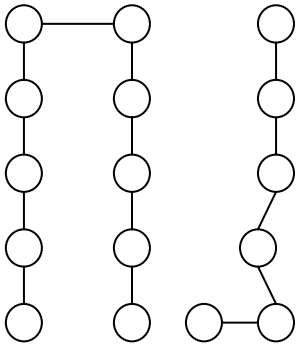


Fig.6-4-1　Step1

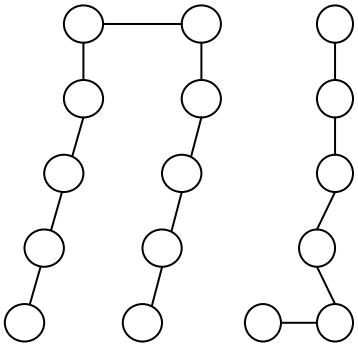


Fig.6-4-2　Step2

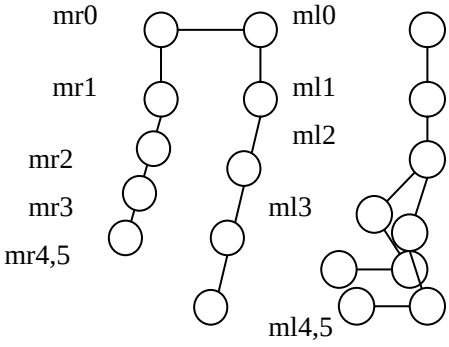


Fig.6-4-3　Step3

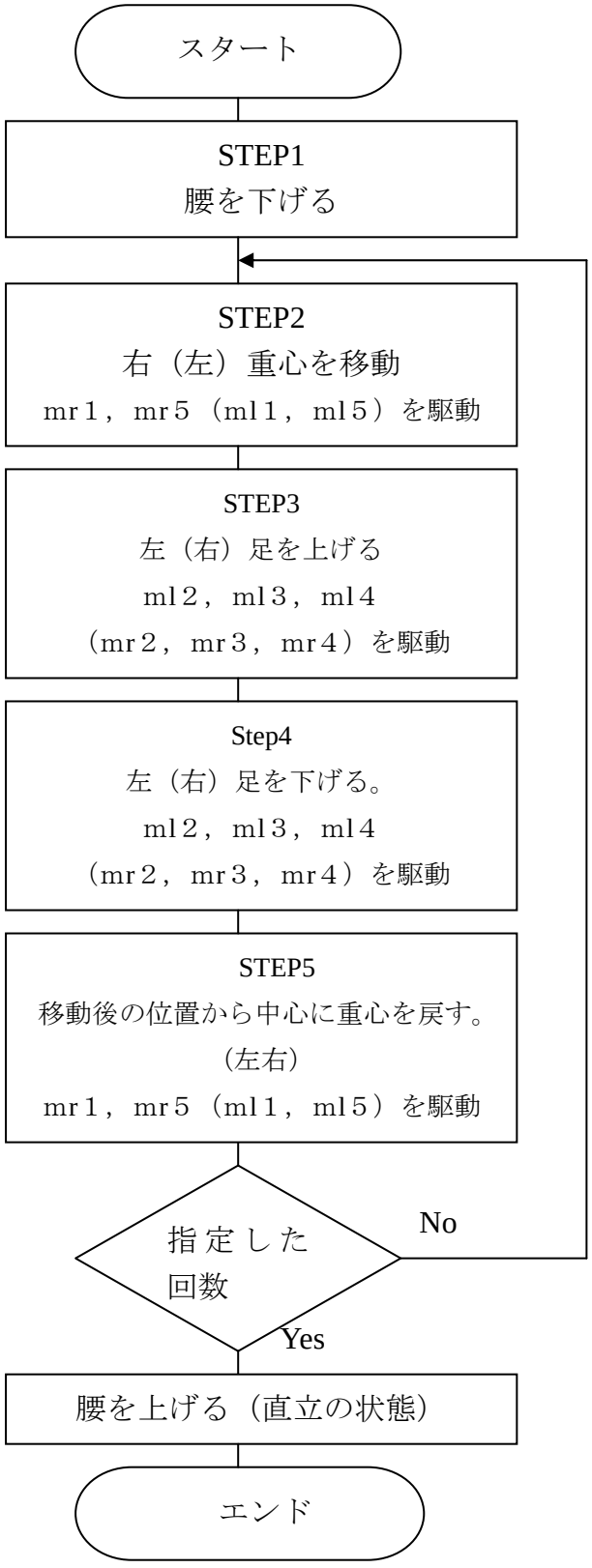


Fig.6-4-4　足踏みのフローチャート

前ページの Fig.6-4-1　Fig.6-4-2　Fig.6-4-3　を繰り返すことによって足踏みが行うことができる.

6-5 歩行

歩行は，重心移動，足上げ足だし，反対側への重心移動，足上げ足だし，重心移動の繰り返しである．ここでの駆動モータはmr1，mr2，mr3，mr4，mr5，ml1，ml2，ml3，ml4，ml5 となっている．歩行のフローチャートを Fig.6-5-5 に示す.

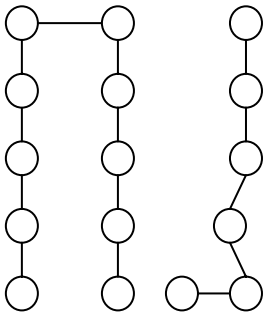


Fig.6-5-1　Step1

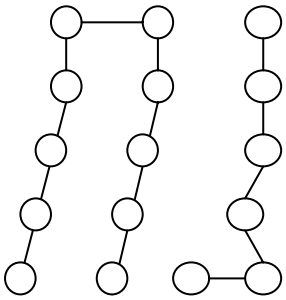


Fig.6-5-2　Step2

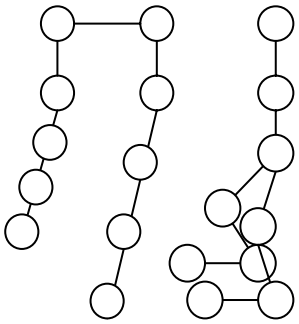


Fig.6-5-3　Step3

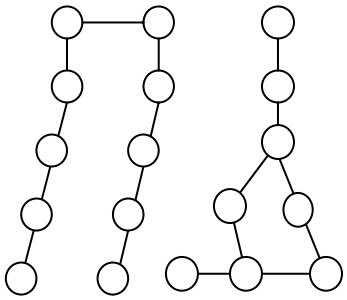


Fig.6-5-4　Step4

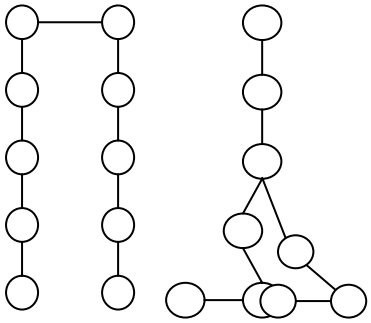


Fig.6-5-5　Step5

この Fig.6-5-1 から Fig.6-455 までを繰り返し，またこの図では左側であるが，右側も繰り返すことによって，歩行することができる.

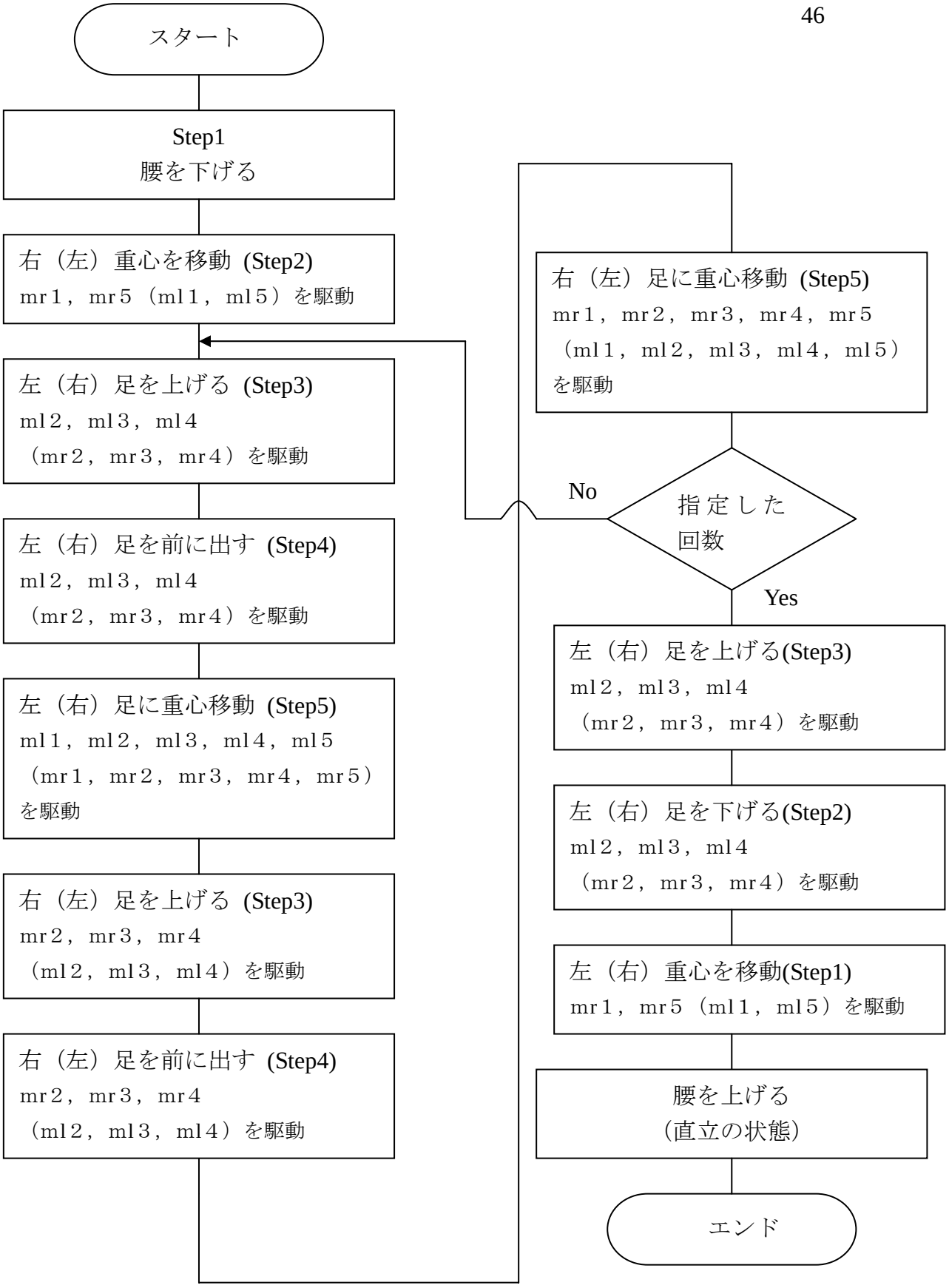


Fig.6-5-6 歩行フローチャート

第七章　結論

本研究において，当初の目的であった二足歩行は実現できた．しかし，我々が成功させた二足歩行は，静歩行のため，あまり速く歩かせることが困難である．静歩行において速く歩かせようとすると，慣性力が作用しバランスを崩し，転倒してしまう．速く歩かせるためには，動歩行を用いる必要があると考えられる．静歩行では，歩行の際に完全に支持脚に重心を乗せる必要がある．しかし，動歩行においては，慣性力を利用することで，重心を支持脚に完全に乗せる前に浮遊脚を前に出すことが可能であるため，重心を支持脚に完全に乗せなくてもいいことになる．また，慣性力の利用の仕方においては，モータへの負荷を減らし，電力の節約にもつながることになる．

機械部品においては現在の状態では問題はないが，今後動歩行や，上半身の制作をする際には，現在使っているパーツでは重いのではないかと考えられる．というのも，1 つのモータに 1 つのブラケットを使用しそのブラケットをネジ止めによって組み立てている．また，モータの振動によってネジがはずれるという出来事もあった，そのため一対型のブラケットで問題がないところにおいては，再度検討が必要であると考えられる．

プログラム No.33

```
//-----------------------------------------------
//メインルーチン
//-----------------------------------------------
int main(void){
	char c;
	int i;
	char txb[10],rxb[10];
	stand();
	sv_cnt=0;		//PWM の切り替えカウンターセット
	SCI1_INIT(br57600,1,txb,sizeof(txb),rxb,sizeof(rxb));	//シリアル（SCI）初期化　57600[bps]
	port_init();		//ポートの初期化
	pwm_init();		//PWM セット
	SCI1_OUT_STRING("¥n<=========================================>¥n") ;
	SCI1_OUT_STRING("¥n<Sample Plogarm for using BTE055>¥n");
	SCI1_OUT_STRING("¥n<	Push Key	a	or	b	or	c	or	d	or	e >¥n");
    while(1){
	  if( SCI1_IN_DATA_CHECK() ){
		c=SCI1_IN_DATA ();
		switch(c){
			case 'a':
				servo_set();//直立
				break;
			case 'b':
				kushin();
				break;
			case 'c':
				jushin();
				break;
			case 'd':
				asiage();
				break;
			case 'e':
				hokou_slowr(30,30,50,30,30); //(腰を落とす量 ko，一歩の時間 tim，歩幅 hohaba，重心移動量 ju,
                                                足を上げる高さ fh）
				break;
			default:break;
		}
```

上記に示すのは，プログラムのメインルーチンである．ここでは，直立状態で，キーボードからの信号を待っている状態である．キーボードの　e　を押すと hokou_slowr(30,30,50,30,30)が選択され，歩行を開始する．

プログラム No.34 より

```
//-----------------------------------------------
//ゆっくりの一歩
//-----------------------------------------------
void hokou_slowr(int ko, int tim, int hohada, int ju, int fh){
        float i;
        int hh,j,kaisu;
        float tt;
        txt=2;
        hh=(hohada)/tim;//速度

        west_down(ko,tim);//          ①腰を落とす                  プログラム No.12 に記載
        l_west(ju,tim);//             ②左に重心移動                プログラム No.10 に記載
          l_west2(ju,tim);//          ③片足をあげるための補正１    プログラム No.10 に記載(左に重心移動 2)
          r_west2(ju,tim);//          ④片足をあげるための補正２    プログラム No.10 に記載(右に重心移動 2)
        f_right(ko,hh,fh,tim,ju);//   ⑤右足を前に                  プログラム No.12
          l_west3(ju,tim);//          ⑥片足をあげるための補正１    プログラム No.11 に記載(左に重心移動 3)
          r_west3(ju,tim);//          ⑦片足をあげるための補正１    プログラム No.10 に記載(右に重心移動 3)
        lr_west_f(ko,ju,hh,tim);//    ⑧　重心を右に移動            プログラム No.13 に記載
        for(kaisu=1;kaisu<=2;kaisu++){  ⑨For 文を使い歩行の回数を決めている
              r_west4(ju,tim);//      ⑩片足をあげるための補正１    プログラム No.11 に記載(右に重心移動 4)
              l_west4(ju,tim);//      ⑪片足をあげるための補正１    プログラム No.11 に記載(左に重心移動 4)
            bm_left2(ko,fh,hh,tim);/  ⑫左足を前へ                  プログラム No.12 に記載
              l_west5(ju,tim);//      ⑬片足をあげるための補正１    プログラム No.11 に記載(左に重心移動 5)
              r_west5(ju,tim);//      ⑭片足をあげるための補正１    プログラム No.11 に記載(右に重心移動 5)
            rl_west_f(ko,ju,hh,tim);// ⑮重心を左に移動              プログラム No.13 に記載
              l_west2(ju,tim);//        片足をあげるための補正１    プログラム No.10 に記載(左に重心移動 2)
              r_west2(ju,tim);//        片足をあげるための補正１    プログラム No.10 に記載(右に重心移動 2)
            f_right2(ko,hh,fh,tim,ju);// ⑯右足を前に                プログラム No.12 に記載
              l_west3(ju,tim);//        片足をあげるための補正１    プログラム No.11 に記載(左に重心移動 3)
              r_west3(ju,tim);//        片足をあげるための補正１    プログラム No.10 に記載(右に重心移動 3)
            lr_west_f(ko,ju,hh,tim);// ⑰重心を右に移動              プログラム No.13 に記載
            }
         r_west4(ju,tim);//             片足をあげるための補正１    プログラム No.11 に記載(右に重心移動 4)
          l_west4(ju,tim);//           片足をあげるための補正１    プログラム No.11 に記載(左に重心移動 4)
        bm_left(ko,fh,hh,tim);//      ⑱左足を前へ                  プログラム No.12 に記載
          l_west5(ju,tim);//           片足をあげるための補正１    プログラム No.11 に記載(左に重心移動 5)
          r_west5(ju,tim);//           片足をあげるための補正１    プログラム No.11 に記載(右に重心移動 5)
        l_westf(ju,tim);//             ⑲重心を戻す                  プログラム No.11 に記載(左に重心移動 f)
        west_up(ko,tim);              ⑳腰を上げる                    プログラム No.12 に記載(腰を戻す)
```

上記に示すのが歩行関数である.

プログラム本論文プ No.1

```
int mmchk(int s_pulse){
   if   (s_pulse>3700){s_pulse=3700;}else{}
   if   (s_pulse<1350){s_pulse=1350;}else{}
   return(s_pulse);
```

ここではモータの限界（±75°）を超えないようにしている。

本論文プログラム No.2

```
PFC.PACRH.BIT.MD16=0; //I/Omode PA16
   PFC.PACRH.BIT.MD17=0; //I/Omode PA17
   PFC.PECR1.BIT.MD15=1;   //MTUmode TIOC4D
   PFC.PECR1.BIT.MD14=1;   //MTUmode TIOC4C
   PFC.PECR1.BIT.MD13=1;   //MTUmode TIOC4B
   PFC.PECR1.BIT.MD12=1;   //MTUmode TIOC4A
   PFC.PECR1.BIT.MD11=1;   //MTUmode TIOC3D
   PFC.PECR1.BIT.MD10=1;   //MTUmode TIOC3C
   PFC.PECR1.BIT.MD9 =1;   //MTUmode TIOC3B
   PFC.PECR1.BIT.MD8 =1;   //MTUmode TIOC3A
   PFC.PECR2.BIT.MD7=1;   //MTUmode TIOC2B
   PFC.PECR2.BIT.MD6=1;   //MTUmode TIOC2A
   PFC.PECR2.BIT.MD5=1;   //MTUmode TIOC1B
   PFC.PECR2.BIT.MD4=1;   //MTUmode TIOC1A
   PFC.PECR2.BIT.MD3=1;   //MTUmode TIOC0D
   PFC.PECR2.BIT.MD2=1;   //MTUmode TIOC0C
   PFC.PECR2.BIT.MD1=1;   //MTUmode TIOC0B
   PFC.PECR2.BIT.MD0=1;   //MTUmode TIOC0A
   //PORT I/O REGISTER
   PFC.PAIORH.WORD=0x00603; //[OUT]PA16,17
   PFC.PEIOR.WORD=0xffff   ;//[OUT]PE0<===>PE15
```

ここでは PFC(ピンファンクションコントローラ)の場所であり、次のページに説明を載せる。

ピンファンクションコントローラ

ピンファンクションコントローラとは、端子の機能を選ぶためと入出力を設定するためのレジスタであり、表.1 のようにそれぞれの端子に対していろいろな機能が割り当てられている。

PWM 信号を出力するためには、マルチファンクションタイマユニット(MTU)を使用する。（本論文に記載）表.1 の機能 2 を選択することによって PWM 信号を TIOC 端子より出力できる。

MTU 関連の入出力端子は 16 本ある。ここで PWM を出力可能なのは、PE ポートに割り当てられた 16 本である。しかし実際には最大 12 本が PWM として使用可能である。これらの端子を MTU に設定するためにはポート E コントロールレジスタ 1・２を使用する。

ポート E コントロールレジスタ 1・2(PECR1,PECR2)は、16 ビットのレジスタでポート E にある 16 本の端子機能を選択する。PECR1 は、ポート E の上位 8 ビットの端子機能を、PECR2 は、ポートＥの下位 8 ビットの端子の機能に割り振られている。

Fig.1 は PECR1 レジスタを示しており、10,11,12,13,14,15 ビットはそれぞれ表.2 のように設定されている。ここで MTU を選択する場合は 01 とする。また 0,2,4,6,8 ビットは表.3 のようになっており、それぞれ 1 を設定すればよいこととなる。

ビット:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	PE15 MD1	PE15 MD0	PE14 MD1	PE14 MD0	PE13 MD1	PE13 MD0	—	PE12 MD	—	PE11 MD	—	PE10 MD	—	PE9 MD	—	PE8 MD
初期値:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fig.1　ポートＥコントロールレジスタ(PECR1)

ビット:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	PE15 IOR	PE14 IOR	PE13 IOR	PE12 IOR	PE11 IOR	PE10 IOR	PE9 IOR	PE8 IOR	PE7 IOR	PE6 IOR	PE5 IOR	PE4 IOR	PE3 IOR	PE2 IOR	PE1 IOR	PE0 IOR
初期値:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fig.2 ポート E･IO レジスタ(PEIOR)

表.1 ピンファンクション

ポート	機能 1 (関連モジュール)	機能 2 (関連モジュール)	機能 3 (関連モジュール)	機能 4 (関連モジュール)
E	PE15 入出力 (ポート)	TIOC4D 入出力 (MTU)	DACK1 出力 (DMCA)	$\overline{IRQOUT}$ 出力 (INTC)
E	PE14 入出力 (ポート)	TIOC4C 入出力 (MTU)	DACK0 出力 (DMCA)	$\overline{AH}$ 出力 (BSC)
E	PE13 入出力 (ポート)	TIOC4B 入出力 (MTU)	$\overline{MRES}$ 出力 (INTC)	
E	PE12 入出力 (ポート)	TIOC4A 入出力 (MTU)		
E	PE11 入出力 (ポート)	TIOC3D 入出力 (MTU)		
E	PE10 入出力 (ポート)	TIOC3C 入出力 (MTU)		
E	PE9 入出力 (ポート)	TIOC3B 入出力 (MTU)		
E	PE8 入出力 (ポート)	TIOC3A 入出力 (MTU)		
E	PE7 入出力 (ポート)	TIOC2B 入出力 (MTU)		
E	PE6 入出力 (ポート)	TIOC2A 入出力 (MTU)		
E	PE5 入出力 (ポート)	TIOC1B 入出力 (MTU)		
E	PE4 入出力 (ポート)	TIOC1A 入出力 (MTU)		
E	PE3 入出力 (ポート)	TIOC0D 入出力 (MTU)	DRAK1 出力 (DMAC)	
E	PE2 入出力 (ポート)	TIOC0C 入出力 (MTU)	$\overline{DREQ1}$ 入力 (DMAC)	
E	PE1 入出力 (ポート)	TIOC0B 入出力 (MTU)	DRAK0 出力 (DMAC)	
E	PE0 入出力 (ポート)	TIOC0A 入出力 (MTU)	$\overline{DREQ0}$ 入力 (DMAC)	

表.2　機能の選択ビット(PE15MD1~PE15MD0)

ビット 15	ビット 14	説明
PE15MD1	PE15MD0	
0	0	汎用入出力(PE15)(初期値)
	1	MTU インプットキャプチャ入力/アウトプットコンペア入力(TIOC4D)
1	0	DMA 要求受付出力(DACK1)(シングルチップでは PE15)
	1	割り込み要求出力(IRQOUT) (シングルチップモードでは予約)

表.3　機能の選択ビット(PE12MD)

ビット 8	説明
PE12MD	
0	汎用入出力(PE12)(初期値)
1	MTU インプットキャプチャ入力/アウトプットコンペア入力（TIOC4A）

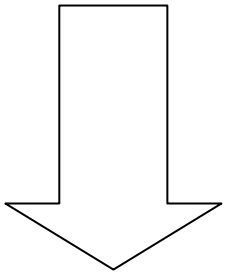
本論文プログラム No.3 より

```
//WDT の設定
   WDT.WRITE.TCSR=0xa53d;   //clock/1024 9.4ms カウントアップ開始
   INTC.IPRH.WORD=0xf000;   //WDT 優先順位＝15
   SetSRReg(0);             //割り込みマスククリア
   }
```

は本論文第 4 章の割り込み処理プログラムにて説明。

本論文プログラム No.4

```
//--------------------------------------
//PWM の設定
//-----------------------------------------
//PWM の初期化
//    SINGLE Ch21<-->Ch24,Ch27<-->Ch30      8cannnel
//    DOUBLE Ch7<-->Ch20,Ch25,Ch,26          16channnel
//-----------------------------------------------
void pwm_init(void){
   PA.DR.WORD.H=0x0000;
//Timer stopuu
   MTU.TSTR.BYTE=0;      //PWM Timer all stop
//ポートへ出力
   PWM.TOER.BYTE=0xff;   //TIOC4D   TIOC4C   TIOC3D   TIOC4B   TIOC4A
TIOC3B
//Timer0 set
   MTU0.TCR.BYTE=0x02;      //not clear TCNT    28MHz/16
   MTU0.TMDR.BYTE=0xc0;   //通常モードでワンショット
```



```
//Timer start
   MTU.TSTR.BIT.CST0=1;    //MTU channel0 start
   MTU.TSTR.BIT.CST1=1;    //MTU channel1 start
   MTU.TSTR.BIT.CST2=1;    //MTU channel2 start
   MTU.TSTR.BIT.CST3=1;    //MTU channel3 start
   MTU.TSTR.BIT.CST4=1;    //MTU channel4 start
  }
```

ここでは、本論文のＭＴＵ（マルチファンクションタイマパルスユニット）にて説明。

プログラム No.5

```
//-----------------------------------------------
// WDT-ITI 割り込みルーチン        defined in 7045.h
//-----------------------------------------------
   //割り込み処理プログラム
   void int_iti(void){
   char a;
     WDT.WRITE.TCSR=0x5a26;//5msec(TCNT  write    maxFF=9.4msec)        //
訂正 10/18
     a=WDT.READ.TCSR.BYTE;   //OVF クリア
     WDT.WRITE.TCSR=0xa53d;   //TCSR への書き込み  max 9.4msec
     pwm_init();          //PWM のセット//時間のカウント
     tim5ms++;//8msec のカウンターインクリメント
```

本論文第 4 章割り込みプログラムにて説明。

本論文プログラム No.6

```
ch[7]=c_ll[0]; ch[8]=c_rl[0];
  ch[24]=c_ll[1]; ch[30]=c_rl[1];
  ch[10]=c_ll[2]; ch[14]=c_rl[2];
  ch[23]=c_ll[3]; ch[29]=c_rl[3];
  ch[22]=c_ll[4]; ch[28]=c_rl[4];
  ch[17]=c_ll[5]; ch[18]=c_rl[5];
```

ここではチャンネルにデータを送っている。

本論文プログラム No.7

```
   //時間のカウント
tim5ms++;tx5++;         //8ms のカウンタインクリメント
//125 回になったら 1 秒単位のカウンタインクリメント
if(tim5ms>125){tim1sec++; tim5ms=0; } else{}
//歩行用カウンタ txt デカウントアップ
if(tx5>txt){txx++; tx5=0;} else{}
PD.DR.WORD.H=tim1sec;
```

ここは、本論文第 4 章時間の作り方で説明。

本論文プログラム No.8

```
//-------------------------------------------------------------
//足をスタート位置から終了位置に移動する．　t:分割数(start<--->end)
//  動作時間=割り込み時間×t
// j:サーボの個数
//-------------------------------------------------------------
void leg_move (int t)
{
        int i,j;
        for(i=0;i<t; i++)    {
              for(j=0;j<12; j++)    {
                    buf_leg[j]=line(j,t,i);
              }
              ax_move(); //サーボにデータをセット
         }
         e2s_leg();     //出発点
}
//
```

t は分割数を表している

buf_leg[j]=line(j,t,i);
buf_leg[]という代数に line 関数にて計算した数値を入力する。

ax_move(); //サーボにデータをセット
各モータに buf_leg[]に入っているデータを入れる。

e2s_leg();   //出発点
end data を変化し start data とする。

本論文プログラム No.9

```
float a,th0,th1,tha,thb,thc;
   a=(2*L1-z)/(2*L1);
   th1=acos(a);
```

・・・①

a は代数，th0～thc は Fig.3 のようになっている．
① th1 にいれる角度の計算を行う．
※ L1 はｍ2～ｍ3，ｍ３～ｍ4 のリンクの長さのことである．今回は L_1＝78〔mm〕として使用している．また Z は時間によって変化する足を上げる高さである．

プログラム No.10

```
a=x/(2*L1-z);
  th0=atan(a);
```

・・・②

② は th0 に入れる角度の計算で x は時間によって変化する Fig.4 であるように足を前に出す量のことである.

プログラム No.11

```
  tha=th0+th1;
  thb=2*th1;
  thc=th1-th0;
```

・・・③

③の tha, thb, thc は Fig.3 に示す角度である。ただし、

th1= θ_1　　th2= θ_2

tha= θ_a　　thb= θ_b

thc= θ_c　　th0= θ_0

に対応する変数である

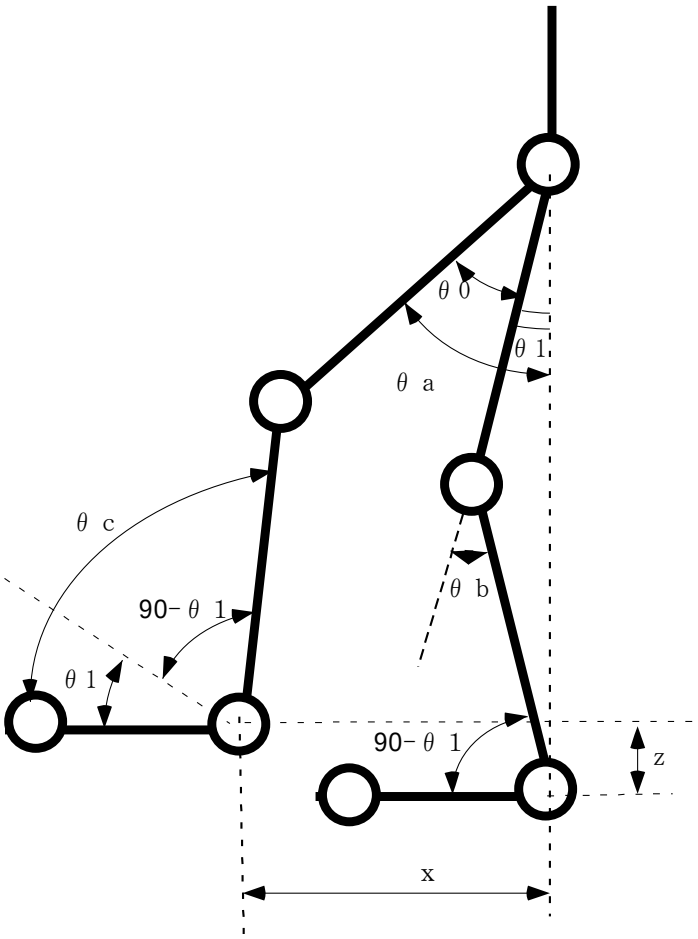


Fig.3 角度を表した簡略図

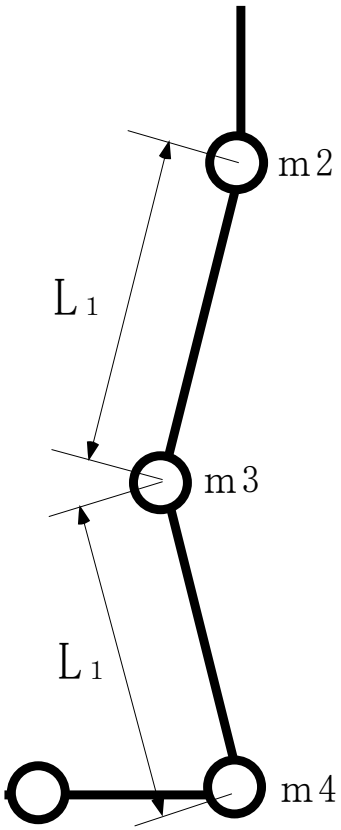


Fig.4　関節間距離を表した簡略図

本論文プログラム No.12

```
a=-tha*(KKK+150)+rln[2];
   c_rl[2]=mmchk(a);
   a=-thb*KKK+rln[3];
   c_rl[3]=mmchk(a);
   a=-thc*(KKK+100)+rln[4];
   c_rl[4]=mmchk(a);
```

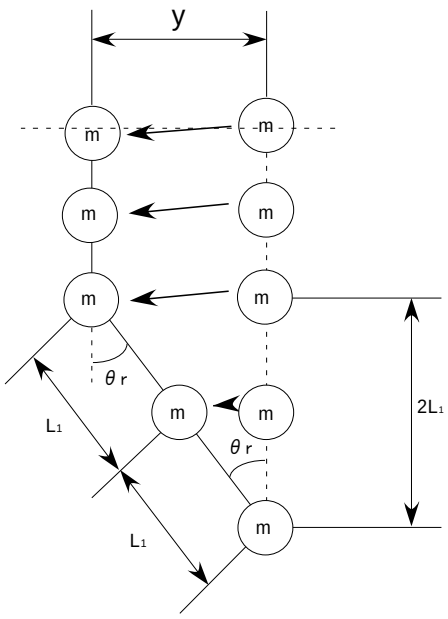
・・・④

④ 計算した tha thb thc の値を c_rl〔〕という代数に入れる．KKK はプログラムで計算された角度がラジアン表示なので、パルスに変換する係数である．KKK の後にある数字は Robocon Magazine【1】より補正量である．
mmchk は，モータが駆動範囲を超えないための関数である．

①～④は右足の制御で（r_x2）であり左足の制御は l_x2 である．

プログラム No.13

```
float a,thr;          ・・・⑤
   a=y/(2*L1)         ・・・⑥
```



⑤は代数の定義
　thr は Fig.5 に示す角度のことである。ただしここで thr= θ r である。

⑥y は Fig.5 に示すように、重心の移動量のことである。またこれは時間によって変化する。

Fig.5　重心を移動した際の図

本論文プログラム No.14
以下は歩行の際の重心を保つためのルーチンである。

```
void y_cntll1sl(int y){
    float a,thr;
    a=y/(2*L1);
    thr=asin(a);
    a=-thr*KKK+2354;
    c_ll[1]=mmchk(a);
    }
```

歩行の際の重心を保つためのルーチン

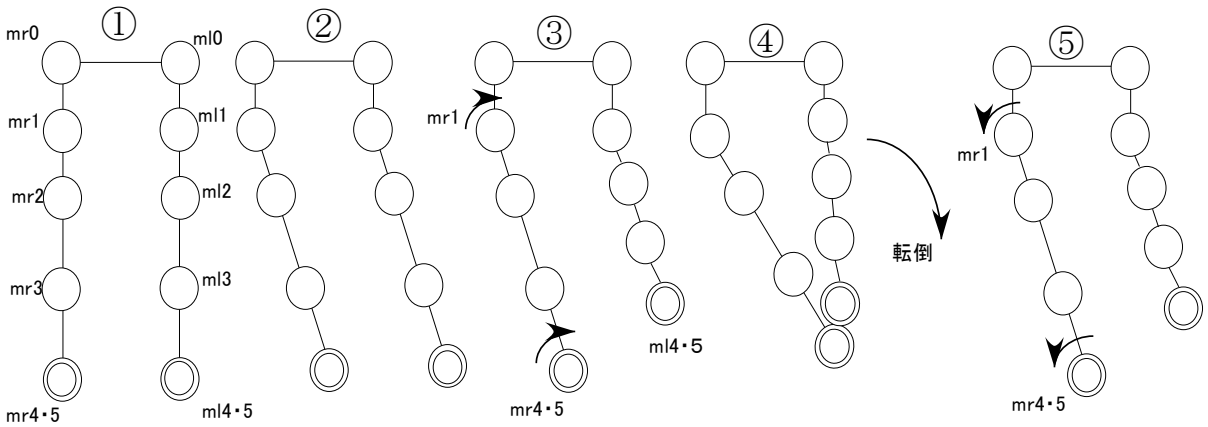


Fig.6　　重心移動

Fig.6 は重心移動のモーションを示すが◎は ml4 のモータと ml5、mr4 と mr5 のモータを示している。

① 腰を落とした状態（mr2, mr3,mr4, ml2, ml3, ml4 のモータが駆動）
② 右に重心移動した状態(mr1, ml1, mr5, ml5 のモータが駆動)
③ 片足をあげたいが左足部分が、ml1 モータの下の方向に力がかかる。また、ml1 mr2 の矢印の方向にトルクがかかる。
④ ③のために、片方の足との接触がおこる。もしくは時計回りに転倒するかのどちらかである。
⑤ Fig.6 において④のためにモータ mr1,mr2 の時計回りに回転する状態にあるので、矢印方向にトルクの分の補正として回転させてやる必要がある。

このようにプログラムが次の歩行の際の重心を保つルーチンである。
Fig.6 の⑤に示した、転倒防止のために改善した重心を保つプログラムを以下に示す。

前ページの続きであり、プログラム No.14

```
void y_cntll1sl(int y){    ・・・⑦
   float a,thr;
   a=y/(2*L1);
   thr=asin(a);
   a=-thr*KKK+2354;    ・・・⑧
   c_ll[1]=mmchk(a);
   }
```

・・・ Ⅰ

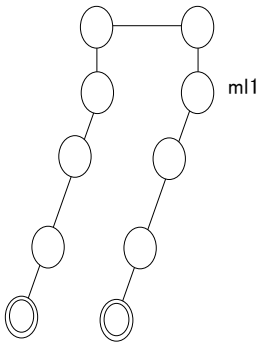


Fig.7　左足への重心移動

⑦ void y_cntll1sl(int y)の
ll1st は、left leg (モータ) 1　start leg の略である。
⑧ における、2354 は、Fig.7 における left leg 1 の値である。

　このプログラムは ml1 モータを 1 個ずつ動かす必要があるので、1 つのモータだけを動かすようにしてある。ここでは y_cntll1sl とあるように left leg モータ 1 （ml1）のモータの制御である。

本論文プログラム No.15 より、

```
void y_cntll5sl(int y){・・・⑨
   float a,thr;
   a=y/(2*L1);
   thr=asin(a);
   a=-thr*KKK+2414;・・・⑩
   c_ll[5]=mmchk(a);
```

・・・ Ⅱ

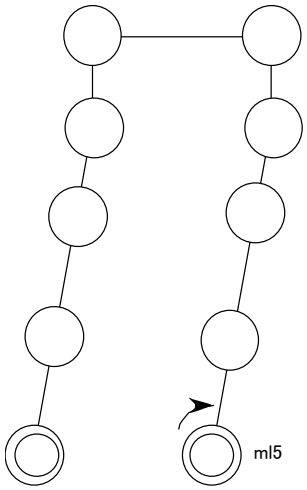


Fig.8 両足傾けている状態の簡略図

⑨ void y_cntll5sl(int y)の ll5sl は
left leg （モータ５）start leg の略である。

この矢印の方向に動かしている.

⑩で示されている 2414 は、Fig.8 の状態での left leg5 の値

本論文プログラム No.16 より、

```
void y_cntll1fl(int y){・・・⑪
   float a,thr;
   a=y/(2*L1);
   thr=asin(a);
   a=-thr*KKK+2519;・・・⑫
   c_ll[1]=mmchk(a);
```

・・・Ⅲ

⑪ void y_cntll1fl(int y)の ll1fl の
left leg （モータ 1）finish leg の略である。

ここでは left leg(number) start leg で片足をあげるために重心をさらに移動させるため, 移動させる分をさらに戻す必要がある．ここではプログラムを表す.

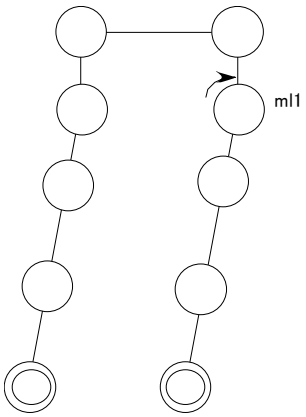


Fig.9 両足を傾けて L1leg の回転

Fig.9 は, 右足をあげて戻した状態であり, Fig.9 ではわからないが y_cntll1sl で矢印の方向に動かしていたため右足が多少浮いている．また浮いている状態になくても完全に足が地面についてはいない状態である．そのためこの動かした分（補正量）を、Fig.9 の矢印と逆方向にモータを駆動させ、もとの状態に戻すことで右足は地面に両足をつけることができる．また y_cntll5sl でも ml5 を動かしているので ml5 も，戻す必要がある.

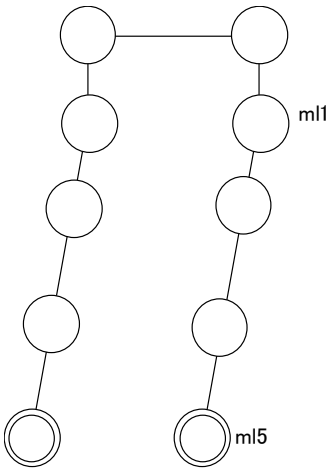


Fig.10 右足をあげて戻した状態

⑫の値は右足をあげてもどした状態の時の,
Fig.10 における、left leg 1（ml1）の値である．この値（2519）から重心移動した後の値（2354）に戻してやる.

本論文プログラム No.17 より

```
void y_cntll5fl(int y){・・・⑬
   float a,thr;
   a=y/(2*L1);
   thr=asin(a);
   a=-thr*KKK+2248;・・・⑭
   c_ll[5]=mmchk(a);
```

・・・Ⅳ

⑬ void y_cntll5fl(int y)
の ll5fl は left leg(モータ５) finish leg の略である。

⑭Fig.10 での left leg 5（二重丸）の値である。

本論文プログラム No.18

```
void y_cntrl1sr(int y){
   float a,thr;
   a=y/(2*L1);
   thr=asin(a);
   a=-thr*KKK+2515;
   c_rl[1]=mmchk(a);
  }
```

・・・Ⅴ～Ⅷ

```
void y_cntrl5fr(int y){
   float a,thr;
   a=y/(2*L1);
   thr=asin(a);
   a=-thr*KKK+2970;
   c_rl[5]=mmchk(a);
   }
```

ここは、left leg の逆であり、つまり right leg のことなので略す。

このⅠ～Ⅷの説明を次のページに記す。

No.14　~　No.18　の数値の説明および測定方法

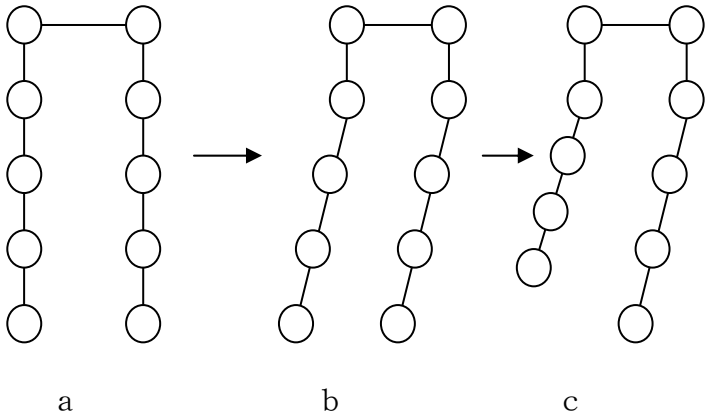


Fig.11 状態変化を表す

Ⅰ～Ⅷのプログラムは歩行，片足をあげる場合にバランスを崩し転倒するのを防ぐためのプログラムである. 例えば Fig.11　a ～ b のように重心移動をしたとする. しかし，この状態において，Fig.1 1　c のように片足をあげると，遊脚の自重によってバランスを崩し，Fig.1 1　c においては反時計回りに転倒をする．それを防ぎ，バランスを保つプログラムである.

まず，なぜこのような数値が必要なのか．それは，この片足をあげる際のバランスの補正をするプログラムを，重心移動を編集して作成したためである．重心移動のプログラムをしたに示す.

重心移動のプログラム

プログラム No.9 に示すように．中心の位置より左右に，指定した値 y だけ移動するものである．プログラム内で指定した値 y を元に，重心移動のために必要な駆動モータ（ml1,ml5,mr1,mr5）の角度計算を行う,計算された値は一度プログラム上で a という代数に入れられ，次に，thr　という代数に入れられる．ここでプログラムを見ると，thr に KKK というものか掛けられ，また＋α(rln[1], rln[5], lln[1],lln[5])されている.

KKK という値は，プログラム内で計算された角度がラジアン表示なので，この角度をパルスに変換するための係数である．また＋α は，この動作をするための基準となる初期値を示している．この初期値を正確に指定しないと，プログラム上では問題がないが，実際にモータに信号が送られた場合問題が生じる．まずこの＋α に 0 を入れたとする．また thr は sin 関数を使用しているため．0 から始まり最高点を通り 0 に戻るプログラムを組んでいる．このことより，＋α を 0 にしてしまうと，パルスは 0 となる．この 0 のパルスがモータに送られたとする．すると 0 はモータの駆動パルス数の範囲外のため，モータに負荷が掛かるなどの問題が生じる．また，腰を少し下げた後に. 重心移動するとき＋α(rln[1], rln[5], lln[1],lln[5])に, 腰を下げた状態での値(rln[1], rln[5], lln[1],lln[5])が入ってないと，問題が生じる．このことを以下に示す

まず．サーボモータ 1 つを例にとり説明をする．

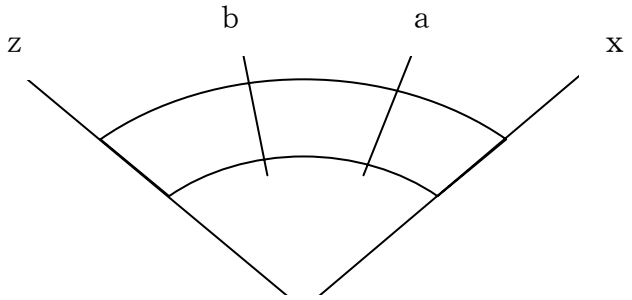


Fig.12　サーボモータ駆動域図

X～Z までをモータの駆動範囲とする．そこで現在 a の位置にモータの角度があるとし，＋α を b とすると，b から行ないたい動作を開始するために，a から b まで急激に変化をした後で，動作を開始する．これによって，モータには慣性力などにより負荷がかかることが考えられる．また，ロボットの場合，いくつものモータを関節として組み立てているため．このようなことが，同時に多数のモータで起こると，バランスを崩し，転倒，につながる．そのため．現在の状態が a であるなら，＋α は a である必要がある．

⑧と⑩は，片足をあげる際にバランスを崩さないために更に重心移動するものである．また，片足をおろす際に，片足をあげるために更に重心移動した分を元に戻す必要がある．それを表すのが，⑫と⑭である．これらの値は共に支持脚である足のみで使用する．

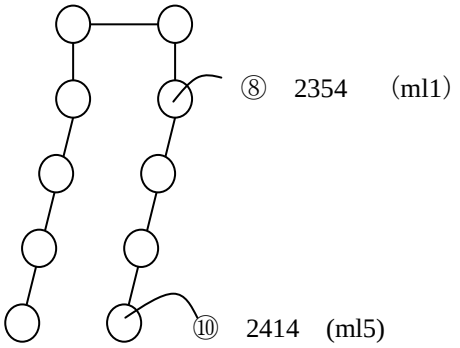


Fig.13　片足をあげる前の値

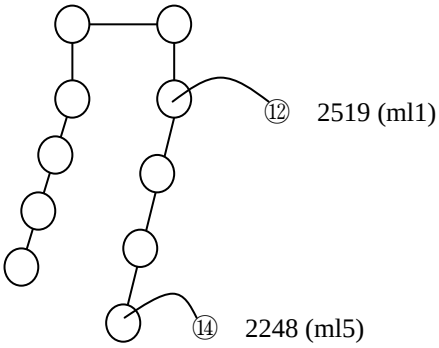


Fig.14　片足をあげている状態での

⑧と⑩の値は，重心移動をしたあとの　ml1 ml5 の値が必要となるそのため，重心移動後の値を測定し，Fig.13　のように決定した．また Fig.14　は片足をあげた際の値を測定する．

測定方法

```
void y_cnt(int y){
    float a,thr;
    a=y/(2*L1);
    thr=asin(a);
    a=-thr*KKK+rln[1];
    c_rl[1]=mmchk(a);
    a=-thr*KKK+rln[5];
    c_rl[5]=mmchk(a);
    a=-thr*KKK+lln[1];
    c_ll[1]=mmchk(a);
    a=-thr*KKK+lln[5];
    c_ll[5]=mmchk(a);
    }
```

プログラム１

```
void y_cnt(int y){
    float a,thr;
    a=y/(2*L1);
    thr=asin(a);
    a=-thr*KKK+rln[1];
    c_rl[1]=mmchk(a);
    a=-thr*KKK+rln[5];
    c_rl[5]=mmchk(a);
    a=-thr*KKK+lln[1];
    SCI1_PRINTF(" PWM=%6d ¥n¥r",(int)a);
    c_ll[1]=mmchk(a);
    a=-thr*KKK+lln[5];
    c_ll[5]=mmchk(a);
    }
```

プログラム２

右に示すのが歩行プログラムである．この歩行プログラムで左に重心移動をしたとする．その際に，測定したい値は ml1 の重心移動を終えたときの値である．そのため，プログラムを実行し ml1 の値が出力されるようにプログラムを変更すればよい

変更したプログラムをプログラム 2 に示す．変更した部分は a=-thr*KKK+lln[1]; の下に SCI1_PRINTF(" PWM=%6d ¥n¥r",(int)a);という関数を加えている．これは a の値を出力する関数である．このプログラムをプログラム 2 に示すよなところに加えることで，大量の数値が出力される．その最後に表示される数値を読み取り．+α の値とすればよい．

このようにして，ほかの 7 つを測定し，プログラムに入力する．またこの値は，重心移動量に比例しているため，重心移動量を変更することにより変化する．そのため重心移動量を変更するたび測定が必要である

本論文プログラム No.19 より

```
void r_west(int ju, int tim){
        float i,tt;
        tx5=0;txx=0;
        while(txx<=tim){・・・⑮
```

⑮txx を tim 以上になるまで while 文の中を繰り返す.

本論文プログラム No.20 より

```
    tt=(float)((txx*3.14/tim)/2); //0->pai/2
        i=sin(tt);
        y_cnt(-i*ju);
```

・・・⑯

⑯グラフにすると

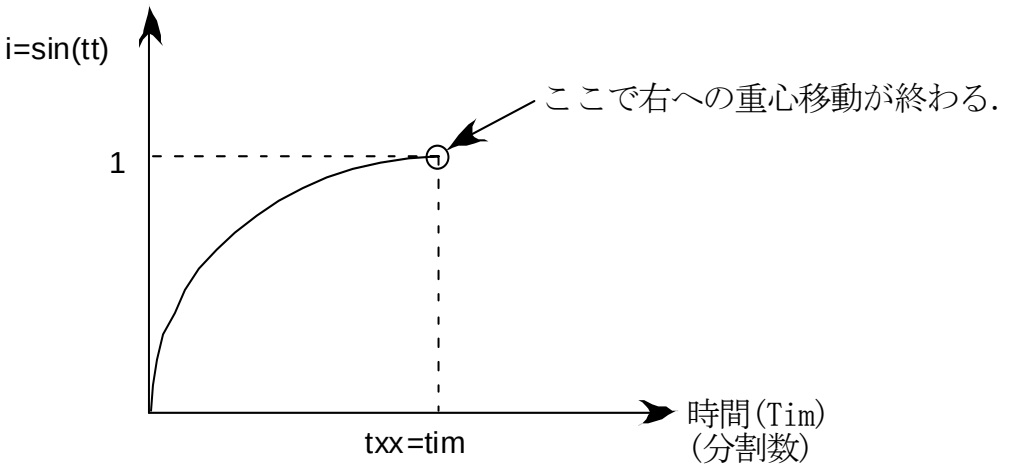


Fig.15　sin カーブ

ここでの i×ju は⑥での y に当たる．また，i は初め 0 であるので腰をおろした状態からはじまる.

またここでの重心移動は腰を落とした状態からスタートするので重心移動した状態のあとにこれを実行してしまうと，一度腰を落とした状態のあとにこれを実行してしまうと，一度腰を落とした状態に急激に変化し，その状態から重心移動を始めてしまう．こうなるとモータに負荷をかけてしまうおそれがあるため注意が必要である.

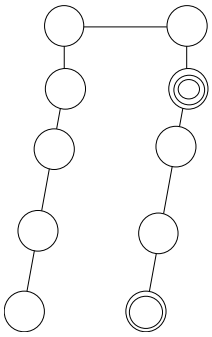
本論文プログラム No.21

```
//左に重心移動
void l_west(int ju, int tim){
        float i,tt;
        tx5=0;txx=0;
        while(txx<=tim){
        tt=(float)((txx*3.14/tim)/2); //0->pai/2
        i=sin(tt);
        y_cnt(i*ju);
        }
}
```

右に重心移動なので省略。

本論文プログラム No.22 の

```
//右に重心移動 2
void r_west2(int ju, int tim){
        float i,tt;
        tx5=0;txx=0;
        while(txx<=tim){
        tt=(float)((txx*3.14/tim)/2); //0->pai/2
        i=sin(tt);
        y_cntll1sl(-i*ju);
        }
}
```



ここでは、左に重心移動させた後に片足をあげるために Fig.16 より Left leg1（三重丸） と　left leg5（二重丸）をさらに回転させる必要がある，ここでは left leg1 を時計方向に回転させるプログラムである.

Fig.16　右に重心移動 2

本論文プログラム No.23 より

```
//左に重心移動 2
void l_west2(int ju, int tim){
        float i,tt;
        tx5=0;txx=0;
        while(txx<=tim){
        tt=(float)((txx*3.14/tim)/2);
//0->pai/2
        i=sin(tt);
        y_cntll5sl(i*ju);
        }
}
```

ここは時計方向に left leg5 を回転させているプログラムである.

本論文プログラム No.24

```
//右に重心移動 3
void r_west3(int ju, int tim){
        float i,tt;
        tx5=0;txx=0;
        while(txx<=tim){
        tt=(float)((txx*3.14/tim)/2);
//0->pai/2
        i=sin(tt);
        y_cntll5fl(-i*ju);
        }
}
```

ここでは左重心移動２で移動させた分を元に戻してやるプログラムである.

プログラム No.25 より

```
//左に重心移動 3
void l_west3(int ju, int tim){
        float i,tt;
        tx5=0;txx=0;
        while(txx<=tim){
        tt=(float)((txx*3.14/tim)/2);
//0->pai/2
        i=sin(tt);
        y_cntll1fl(i*ju);
```

ここでは右重心移動２で移動させた分を元に戻すプログラムである。

プログラム No.26

```
//右に重心移動 4
void r_west4(int ju, int tim){
        float i,tt;
        tx5=0;txx=0;
        while(txx<=tim){
        tt=(float)((txx*3.14/tim)/2);
```

```
//左に重心移動 5
void l_west5(int ju, int tim){
        float i,tt;
        tx5=0;txx=0;
        while(txx<=tim){
        tt=(float)((txx*3.14/tim)/2);
//0->pai/2
        i=sin(tt);
        y_cntrl5fr(i*ju);
```

　これらは右・左重心移動２、３の繰り返しである。

プログラム No.27

```
//右に重心移動 f
void r_westf(int ju, int tim){
        float i,tt;
        tx5=0;txx=0;
        while(txx<=tim){
        tt=(float)((txx*3.14/tim)/2);
//0->pai/2
        i=sin(tt);
        y_cnt(ju-i*ju);・・・⑰
        }
}
//左に重心移動 f
void l_westf(int ju, int tim){
        float i,tt;
        tx5=0;txx=0;
        while(txx<=tim){
        tt=(float)((txx*3.14/tim)/2);
//0->pai/2
        i=sin(tt);
        y_cnt(-ju+i*ju);
        }
}
```

重心移動した状態から，腰を落とした状態にもっていくプログラムである．
右に重心移動した f は，左に重心移動をした後に使用する．左に重心移動 f は右に重心移動をした後に使用する．

⑰ju-i*ju
右への重心移動であれば-I*ju でいいのだが，ここでは ju-I*ju となっている．この ju は初めに左に ju の量だけ，移動してあるためあらかじめ移動してある量から戻すプログラムとなっている．よってこの ju がない状態であると一度腰を下ろした状態に急激に変化しそこからさらに右に重心移動してしまうことになる．
つまり、Fig.17 の左に重心移動している．
この状態から中心に戻すためには、
ju-i×ju
となる。

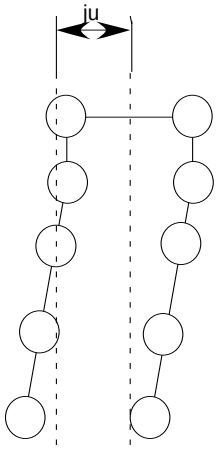


Fig.17 左に重心移動 f

本論文プログラム No.28 より

```
//右足を前に
void f_right(int ko, int hh,int fh, int tim,int ju){
        float tt,i,v;
        tx5=0;txx=0;
        while(txx<=tim){
        tt=(float)(txx*3.14/(tim));
        i=sin(tt);
        v=i;
          i=i*fh;       //足を上げる高さ
          r_xz(hh*txx,(int)i+ko);
        }
}
```

これは右足を前に出すプログラムである。ただし腰を落とした状態から始まるプログラムであるため、初めの一歩に使う。

本論文プログラム No.29

```
//右足を前に 2
void f_right2(int ko, int hh,int fh, int tim,int ju){
        float tt,i,v;
        tx5=0;txx=0;
        while(txx<=tim){
        tt=(float)(txx*3.14/(tim));
        i=sin(tt);
        v=i;
          i=i*fh;       //足を上げる高さ
          r_xz(hh*2*txx-hh*tim,(int)i+ko);・・・⑱
        }
}
```

⑱右足を前に出すプログラムではあるが、上のプログラムとは少し異なっており、後ろ足を前に出すプログラムである。

本論文プログラム No.30

```
//右足を前に 3
void f_right3(int ko, int hh,int fh, int tim,int ju){
        float tt,i,v;
        tx5=0;txx=0;
        while(txx<=tim){
        tt=(float)(txx*3.14/(tim));
        i=sin(tt);
        v=i;
          i=i*fh;       //足を上げる高さ
          r_xz(hh*txx-hh*tim,(int)i+ko);
        }
}
```

これは後ろ足を元に戻す（初めの状態）に戻すプログラムである。

本論文プログラム No.31

```
//左足を前へ
void bm_left(int ko, int fh,int hh, int tim){
        float i,tt;     //
        tx5=0;txx=0;
        while(txx<=tim){
```



```
//左足を前へ 3
void bm_left3(int ko, int fh,int hh, int tim){
        float i,tt;     //
        tx5=0;txx=0;
        while(txx<=tim){
        tt=(float)(txx*3.14/(tim));
                //
        i=sin(tt);
          i=i*fh;       //足を上げる高さ
          l_xz(hh*txx,(int)i+ko);
        }
```

　これは、前に記したように右足と同様のものである。

プログラム No.32

```
//重心を右に移動
void lr_west_f(int ko, int ju, int hh, int tim){
	float i,tt;
	tx5=0;txx=0;
	while(txx<=3*tim){
	l_xz(-hh*txx/3,ko);
	r_xz((hh*tim-hh*txx/3),ko);

	tt=(float)(txx*3.14/(3*tim));
//0->pai/2
	i=cos(tt);
		y_cnt(i*ju);
	}
}
//重心を左に移動
void rl_west_f(int ko, int ju, int hh, int tim){
	float i,tt;
	tx5=0;txx=0;
	while(txx<=3*tim){
	l_xz((hh*tim-hh*txx/3),ko);
	r_xz(-hh*txx/3,ko);

	tt=(float)(txx*3.14/(3*tim));
//0->pai/2
	i=cos(tt);
		y_cnt(-i*ju);
	}
}
```

Fig.18 のように右足を前に出すと、でた状態では重心が後ろ足にあるため、前足に重心を移動する必要がある。

ロボットの進行方向を y 軸でそれに水平で垂直方向に x 軸をとると a の方向に重心移動を行うことになる。（Fig19）

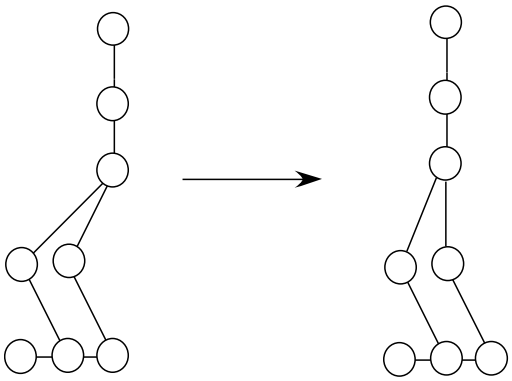


Fig.18　重心の移動

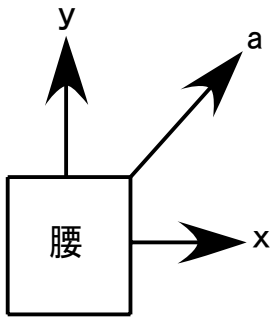


Fig.19 ロボットの重心移動方向

本論文プログラム No.33 より

```
l_west2(ju,tim);//
        r_west2(ju,tim);//
```

これは足を上げた状態でトルクがかかるため、このトルクに負けないような力を与えてやる必要がある。そこでイメージ的に Fig.20 のようになるように回転力を与えてやる。

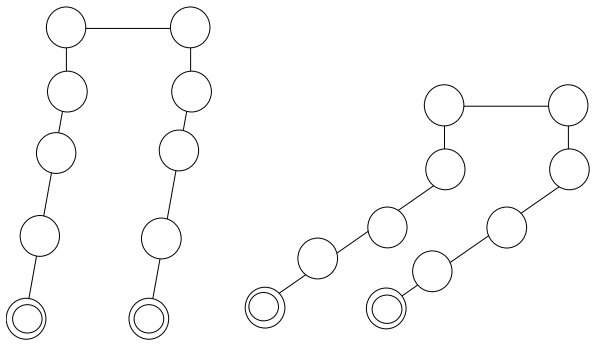


Fig.20 重心移動

本論文プログラム No.33 より

```
l_west3(ju,tim);//
r_west3(ju,tim);//
```

これは Fig.20 の重心を元に戻すものである。つまり、左から右に移動するものである。