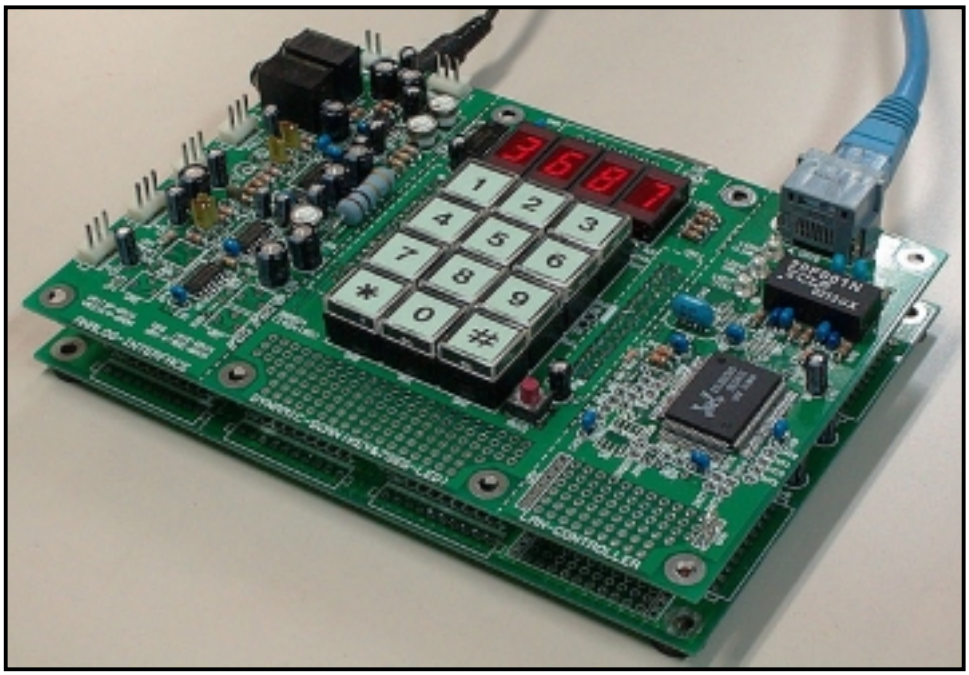
ーTK-3687オプションー

I/Fトレーニングユニット ユーザーズマニュアル

〈基礎編〉



Version1.02

目次

(株)東洋リンクス

はじめに

I/Fトレーニングユニットは，マイコンボード TK-3687 と接続してマイコンのインターフェースを学習するために作られた拡張ボードです。インターフェースとして比較的に身近なテーマを取り上げ，中級から上級レベルのハード・ソフトの学習を行ないます。TK-3687 とは基板間コネクタで接続，TK-3687 とスタッキングします。ハード的には 3 つのブロックに分かれており，それぞれ独立して動作させることが可能です。

部品は全て実装済みです。マイコン側でコネクタの取り付け等がありますが，すぐに動作させることが可能です。回路図を掲載していますので，プログラムの学習の際に見比べる事で，よりマイコンのハード・ソフトの関連性が理解できるのではないかと思います。

本マニュアルは基礎編ということもあり，本文中ではあまり詳しい解説がされていません。学習の中心はソースプログラムを読み，それを理解することになります。したがって最初は大変かもしれませんが，ソースの中のコメントをヒントにしていけば，次第に部分部分で何をやっているのかわかってくると思います。

プログラムで使用した言語は‘C’です。最近はワンチップマイコンでも‘C’が使われるようになりました。ルネサステクノロジーから無償版コンパイラ‘HEW’が公開されています。組み込み用途における‘C’によるプログラムを是非体験されてください。きっと，アセンブラにはない利点と，‘C’の多少の限界が理解でき，プログラミング技術の幅を広げる事ができると思います。

プログラムのダウンロード

付属の CD-ROM に本学習キットで使用するプログラムが含まれています。第 1 章から第 3 章までは TK-3687 に書かれているモニタプログラム，ハイパーH8 で RAM にダウンロードします。ダウンロード方法については，同じ CD-ROM に含まれている‘TK-3687 ユーザーズマニュアル C 言語版’をご覧下さい。

第 4 章「LAN によるハードウェアの制御」についてはプログラムエリアとワークファイルのサイズの関係で H8/3687 のフラッシュメモリに書き込む必要があります。書込みにはルネサステクノロジーが無償配布している‘FDT’を使用します。但し，ハイパーH8 を引き続き使用するため，ハイパーH8 と LAN のプログラムを一緒に書き込みます。‘FDT’によるプログラムのダウンロード方法，及び，ハイパーH8 と一緒に書き込む方法は，同じ CD-ROM に含まれている‘TK-3687 ユーザーズマニュアル C 言語版’をご覧下さい。

なお，ダウンロードするのは S タイプファイルで，CD-ROM の下記の場所にあります。（次頁）

CD-ROM プログラム一覧（本マニュアルに関係するものを抜粋）

プログラムの実行方法

第 1 章から第 3 章までは“ハイパーH8”でダウンロードします。ダウンロードすると自動的に PC にスタート番地がセットされるので，ダウンロード後，次のコマンドを入力して実行します。

```
H8>G  Enter
```

第 4 章はあらかじめフラッシュメモリに書き込まれたプログラムを実行するので，次のコマンドを入力して 7400h 番地から実行します。

```
H8>G7400  Enter
```

TK-3687 の設定

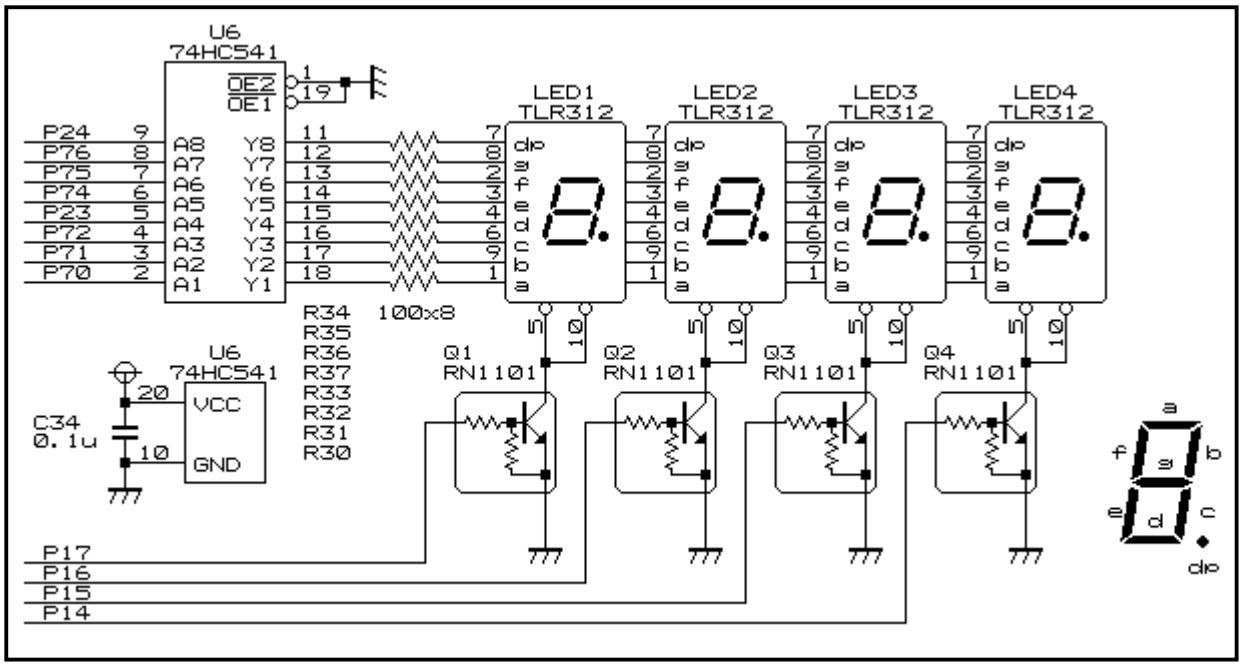
I/F トレーニングユニットを使うにあたり，TK-3687 を次のように設定して下さい。

① TK-3687 の JP4，JP5，JP6 をラッピングワイヤなどでショートする。
② TK-3687 の CN16 と CN17 に付属の 30 ピンコネクタを実装する。（要ハンダ付け，方向に注意）
③ TK-3687 の CN16 の 19 番ピンと CN18 の 12 番ピンをラッピングワイヤで接続する。（要ハンダ付け）

第1章 ダイナミックスキャン

1. ダイナミック表示

1-1 回路図(表示部抜粋)

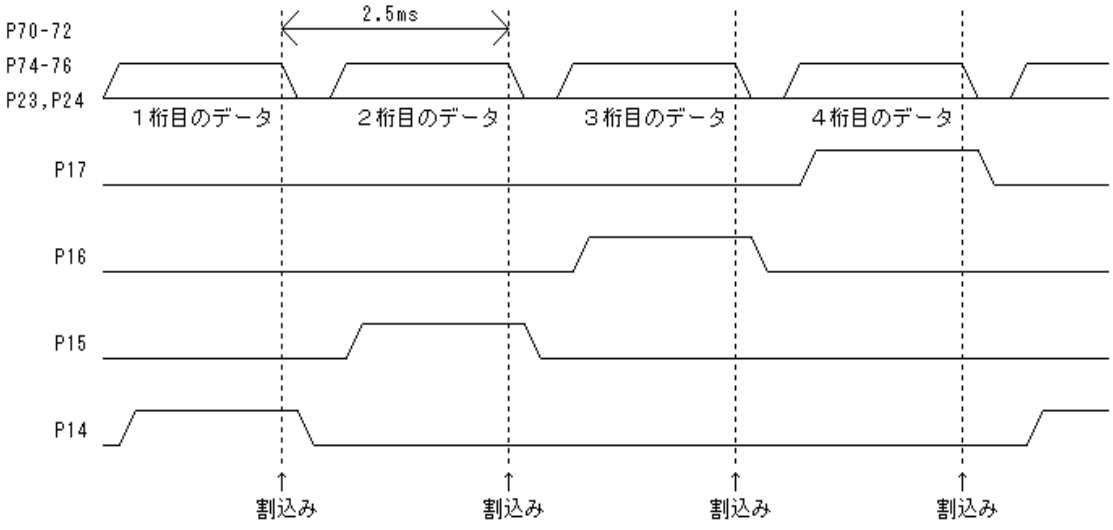


1-2 タイミングチャート

回路図からわかるように, P14～17 の信号を“High”にするとトランジスタが ON の状態になりコモン端子は“Low”になります。その時 P70～72, P74～76, 及び P23, P24 の点灯させたいビットを“High”にすると LED に電流が流れて点灯します。ある一つの桁を表示している間は他の桁を消灯し, それを 4 桁分繰り返し行います。人間の目の残像現象を利用し, 消えているのがわからないくらいの速さで切替えていけば, 全ての桁が同時に点灯しているように見せかける事ができます。これをダイナミック表示と呼びます。

今回のプログラムでは切替えるタイミングをタイマ割り込みで作っています。タイマ B1 は約 2. 5ms で割り込みをかけるようにイニシャライズしています。

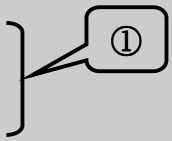
これを, タイミングチャートで表すと右のようになります。このタイミングチャートをもとにプログラムを作ります。



1-3 ソースリスト(抜粋)

タイマ B1 割り込みの中で, LED スキャンルーチン, ‘scan()’関数をコールします。タイミングチャートに従い, 表示を消去した後, 次の桁の表示データを出力, 最後にスキャンラインをオンして表示します。‘scan()’関数のリストは次の通りです。

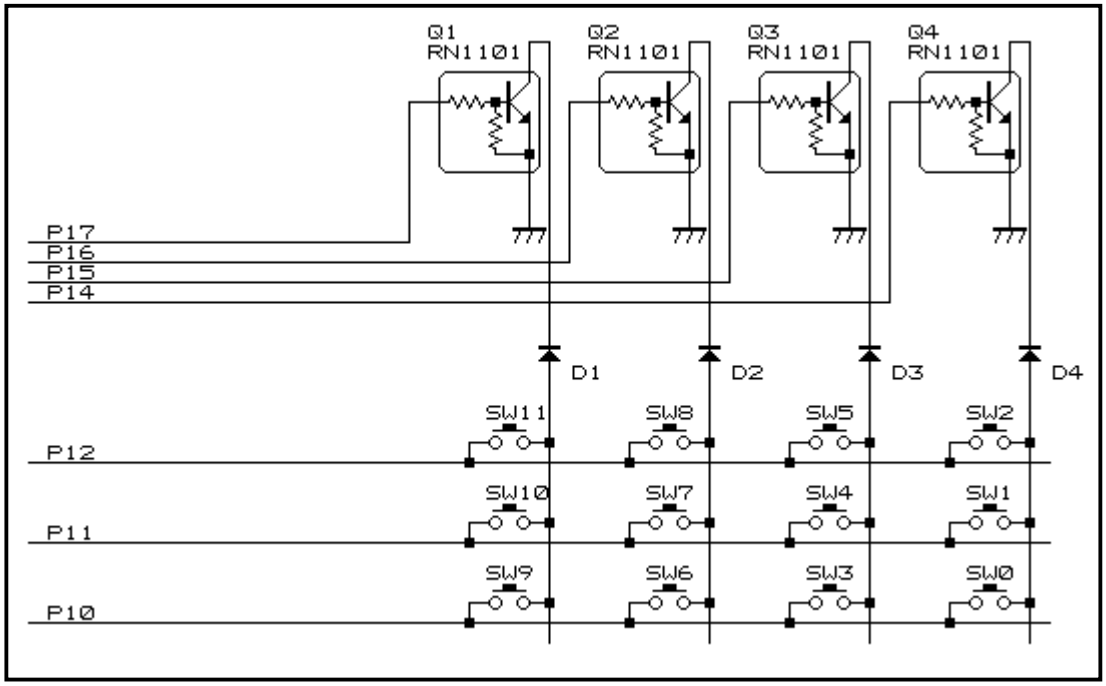
```
/*****************************************************************************
    LED スキャン
*****************************************************************************/
void scan(void)
{
    //表示消去
    P_PORT.PDR7.BYTE = 0x00;
    P_PORT.PDR2.BYTE = P_PORT.PDR2.BYTE & 0xe7;

    //スキャンラインオールディセーブル
    P_PORT.PDR1.BYTE = P_PORT.PDR1.BYTE & 0x0f;

    //スキャンカウンタの更新
    ScanPnt++; if (ScanPnt>(KETA-1)) ScanPnt=0;

    //次のセグメントを表示する
    P_PORT.PDR7.BYTE = SegBuf[ScanPnt] & 0x77;
    P_PORT.PDR2.BIT.P23 = ((SegBuf[ScanPnt]&0x08) ? 1 : 0);
    P_PORT.PDR2.BIT.P24 = ((SegBuf[ScanPnt]&0x80) ? 1 : 0);
    switch (ScanPnt){
        case 0:
            P_PORT.PDR1.BIT.P14 = 1; break;
        case 1:
            P_PORT.PDR1.BIT.P15 = 1; break;
        case 2:
            P_PORT.PDR1.BIT.P16 = 1; break;
        case 3:
            P_PORT.PDR1.BIT.P17 = 1; break;
    }
}
```



セグメントデータは①で出力しています。回路図から判るとおり, データのビット 0～2, ビット 4～6 はそれぞれ P70～72, P74～76 に, ビット 3 は P23, ビット 7 は P24 に出力します。

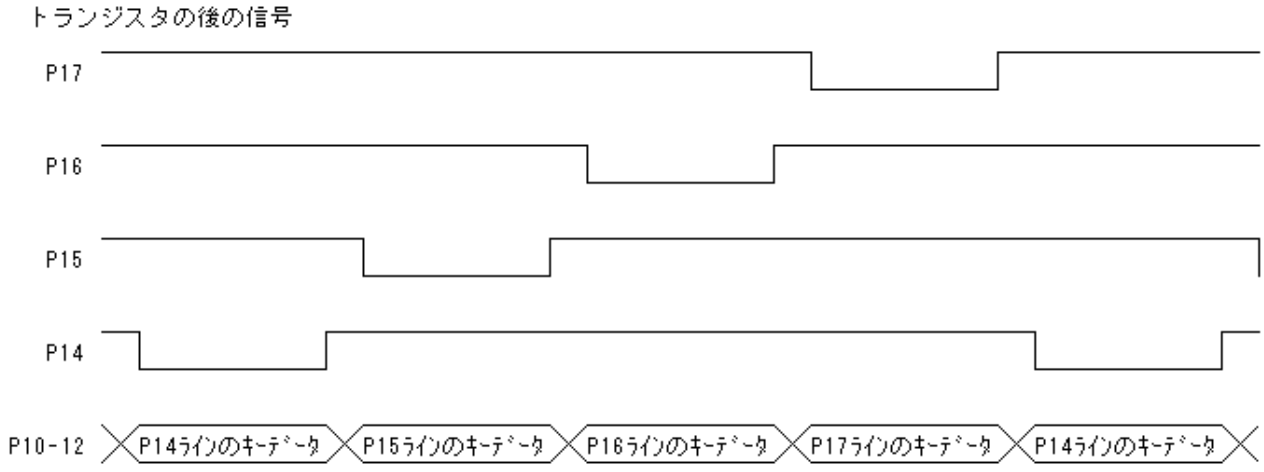
2. マトリックスキー

2-1 回路図(キー部抜粋)



2-2 タイミングチャート

P14～17 の内 1 つのスキャンラインを Low にしてキーを読み込み，1 ライン読み込んだら次のスキャンラインを Low に，これをスキャンライン分繰り返します。P10～12 は CPU 側でプルアップされていますので，スイッチが押されているときに Low になります。実際のプログラムではチャタリングを除去するために 2 回スキャンして，一致したときだけ押されたと判定します。したがって，10ms 以下の場合はノイズとみなされ，認識されません。



なお，キーのスキャンと LED のスキャンは共通にしています。(P. 14 回路図参照)

2-3 ソースリスト(抜粋)

キーのスキャンは LED のスキャンと共通のため，タイマ B1 割り込みの中でキースキャンルーチン，‘keyin()’関数をコールします。チャタリング除去のため 2 回スキャンして一致した時だけキー入力ありと見なします。‘keyin()’関数のリストは次の通りです。

LED とスキャンラインを共通としているため，この関数ではキーデータの入力処理を行い，スキャンラインの更新は‘scan()’関数内で行なっています。

```
/************************************************************************
    キースキャン
************************************************************************/
void keyin(void)
{
    int i;
    int key;      //押されているキーの中で一番若いキー番号
    int push;     //0:何も押されていない 1:キーが押された
    int equal;    //OK:1st=2nd NG:1st!=2nd

    switch (KeyReadFlag){
        // 1st リード ------------------------------------------------
        case 0:
            Key1stBuf[ScanPnt] = ~P_PORT.PDR1.BYTE & 0x07;
            if (ScanPnt>=(KETA-1)){          // スキャン終了
                KeyReadFlag = 1;
            }
            break;
        // 2nd リード ------------------------------------------------
        case 1:
            Key2ndBuf[ScanPnt] = ~P_PORT.PDR1.BYTE & 0x07;
            if (ScanPnt>=(KETA-1)){          // スキャン終了
                KeyReadFlag = 0;
                equal = OK; key = 0; push = 0;

                // 1st=2nd かチェック
                for (i=0;i<KETA;i++){
                    if (Key1stBuf[i]!=Key2ndBuf[i]){
                        equal = NG; break;      // 同じではなかった
                    }
                }

                // 押されたキー番号を得る，キー番号が若い方が優先
                if (equal==OK){         // 1st=2ndのとき
                    for (i=KETA-1;i>=0;i--){
                        KeyBuf[i] = Key2ndBuf[i];
                        if      ((KeyBuf[i]&0x01)==0x01){
                            key = i*3+0; push = 1;
                        }
                        else if ((KeyBuf[i]&0x02)==0x02){
                            key = i*3+1; push = 1;
                        }
                        else if ((KeyBuf[i]&0x04)==0x04){
```

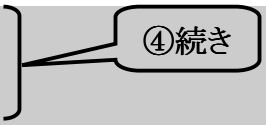
①

②

③

④

```
                        key = i*3+2; push = 1;
                    }
                }

                if (push==1){     // 押されている
                    if (KeyNoOld!=key){    // 前回と違うキーが押されている
                        KeyNo = key; KeyNoOld = key;
                        KeyFlag = 1;
                    }
                }
                else{             // 何も押されていない
                    KeyNoOld = -1;
                }
            }
        }
        break;
    }
}
```



プログラムは大きく分けると①～⑤に分かれており，それぞれ，

① ファーストリード。各スキャンのデータを読み，全スキャンが終了したら‘KeyReadFlag=1’にする。

② セカンドリード。各スキャンのデータを読み，全スキャンが終了したら‘KeyReadFlag=0’にし，キー判定に移る。

③ ファーストリードとセカンドリードを比較し，一致しているか確認する。

④ 押されているキーのうち，もっとも若いキー番号を取得する。

⑤ 前回のスキャンで得たキー番号と比較し，違うキーが押されているときに‘KeyNo’にキー番号をセット，‘KeyFlag=1’にする。

となります。

3. ソースリスト

ダイナミックスキャンとマトリックスキーの入力を合わせたプログラムの全ソースリストを示します。メインルーチンでは，キーが押されたら表示を左にシフトし，最右桁に押されたキーの値を表示します。なお，‘*’キーは‘A’，‘＃’キーは‘B’と表示します。

```
/************************************************************************/
/*                                                                      */
/*  TK-3687用 IFトレーニングユニット                                    */
/*  ダイナミックスキャン                                                */
/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */
/*  FILE        :dscan.c                                                */
/*  DATE        :Tue, Jan 20, 2004                                      */
/*  DESCRIPTION :Main Program                                           */
/*  CPU TYPE    :H8/3687                                                */
/*  BOARD       :TK-3687(B6081) & I/F-Unit(B6084)                       */
/*                                                                      */
/*  This file is programed by TOYO-LINX Co.,Ltd. / yKikuchi             */
/*                                                                      */
/************************************************************************/

/************************************************************************
    履歴
************************************************************************/
/*
2004-01-20 : 作成開始
*/


/************************************************************************
    インクルードファイル
************************************************************************/
#include  <machine.h>
#include  "iodefine.h"


/************************************************************************
    定数の定義（直接指定）
************************************************************************/
// 7セグメントLEDに関係する定数 ----------------------------------------
#define       KETA          4         //7セグメントLEDの桁数

// その他 --------------------------------------------------------------
#define       OK                0         //戻り値
#define       NG                -1        //戻り値


/************************************************************************
    定数の定義（ROM）
************************************************************************/
// 7セグメントデータ変換テーブル ---------------------------------------
const unsigned charSegTable[16] = {0x3f,0x06,0x5b,0x4f,   //0,1,2,3
                                   0x66,0x6d,0x7d,0x07,   //4,5,6,7
                                   0x7f,0x67,0x77,0x7c,   //8,9,A,B
```

```
                                0x39,0x5e,0x79,0x71};  //C,D,E,F

// キー番号変換テーブル ----------------------------------------------
const unsigned charKeyTable[12] = {0x0a,0x00,0x0b,
                                0x07,0x08,0x09,
                                0x04,0x05,0x06,
                                0x01,0x02,0x03};


/*************************************************************************
    スタティック変数の定義とイニシャライズ
    ここでイニシャライズしていない変数はプログラム中で行なっている
*************************************************************************/
// 7セグメントLEDスキャンに関する変数 ---------------------------------
unsigned char  SegBuf[4] = {0x07  //7セグメントLEDバッファ
                           ,0x7f  //（初期表示）
                           ,0x7d
                           ,0x4f};
unsigned char  ScanPnt = 0;       //どのスキャンラインか
                                  //（キースキャンと兼用）

// キースキャンに関する変数 -------------------------------------------
unsigned char  Key1stBuf[4];      //キー1stリード
unsigned char  Key2ndBuf[4];      //キー2ndリード
unsigned char  KeyBuf[4];         //1st=2nd になったキーデータ
unsigned char  KeyReadFlag = 0;   //キーリードフラグ
                                  // 0:1stリード
                                  // 1:2ndリード
int            KeyNo = 0;         //キー番号
int            KeyNoOld = -1;     //前回のキー番号
unsigned char  KeyFlag = 0;       //キー入力フラグ
                                  // 1:キー入力あり


/*************************************************************************
    関数の定義
*************************************************************************/
void           init_int(void);
void           init_port(void);
void           init_tmrb1(void);
void           intprog_tmrb1(void);
void           keyin(void);
void           main(void);
void           scan(void);


/*************************************************************************
    メインプログラム
*************************************************************************/
void main(void)
{
    int i;

//----- イニシャライズ ------------------------------------------------
    init_tmrb1();
    init_port();
    init_int();
```

```
    set_imask_ccr(0);  //割り込みイネーブル

//----- メインループ ------------------------------------------------
    while(1){
        if (KeyFlag==1){   // キー入力あり
            KeyFlag = 0;
            for (i=KETA-1; i>0; i--){    // 表示左シフト
                SegBuf[i] = SegBuf[i-1];
            }
            SegBuf[0] = SegTable[KeyTable[KeyNo]]; // 右端に新データを表示
        }
    }
//-----------------------------------------------------------------
}


/*************************************************************************
    キー&LED スキャン（タイマB1割り込み）
*************************************************************************/
#pragma regsave (intprog_tmrb1)
void intprog_tmrb1(void)
{
    P_INT.IRR2.BIT.IRRTB1 = 0;   // タイマB1割込み要求フラグクリア
    keyin(); //キースキャン
    scan();  //7セグメントLEDスキャン，スキャンラインの制御はここ
}


/*************************************************************************
    キースキャン
*************************************************************************/
void keyin(void)
{
    int i;
    int key; //押されているキーの中で一番若いキー番号
    int push; //0:何も押されていない 1:キーが押された
    int equal;    //OK:1st=2nd NG:1st!=2nd

    switch (KeyReadFlag){
        // 1st リード ---------------------------------------------
        case 0:
            Key1stBuf[ScanPnt] = ~P_PORT.PDR1.BYTE & 0x07;
            if (ScanPnt>=(KETA-1)){      // スキャン終了
                KeyReadFlag = 1;
            }
            break;
        // 2nd リード ---------------------------------------------
        case 1:
            Key2ndBuf[ScanPnt] = ~P_PORT.PDR1.BYTE & 0x07;
            if (ScanPnt>=(KETA-1)){      // スキャン終了
                KeyReadFlag = 0;
                equal = OK; key = 0; push = 0;

                // 1st=2nd かチェック
                for (i=0;i<KETA;i++){
                    if (Key1stBuf[i]!=Key2ndBuf[i]){
                        equal = NG; break; // 同じではなかった
                    }
```

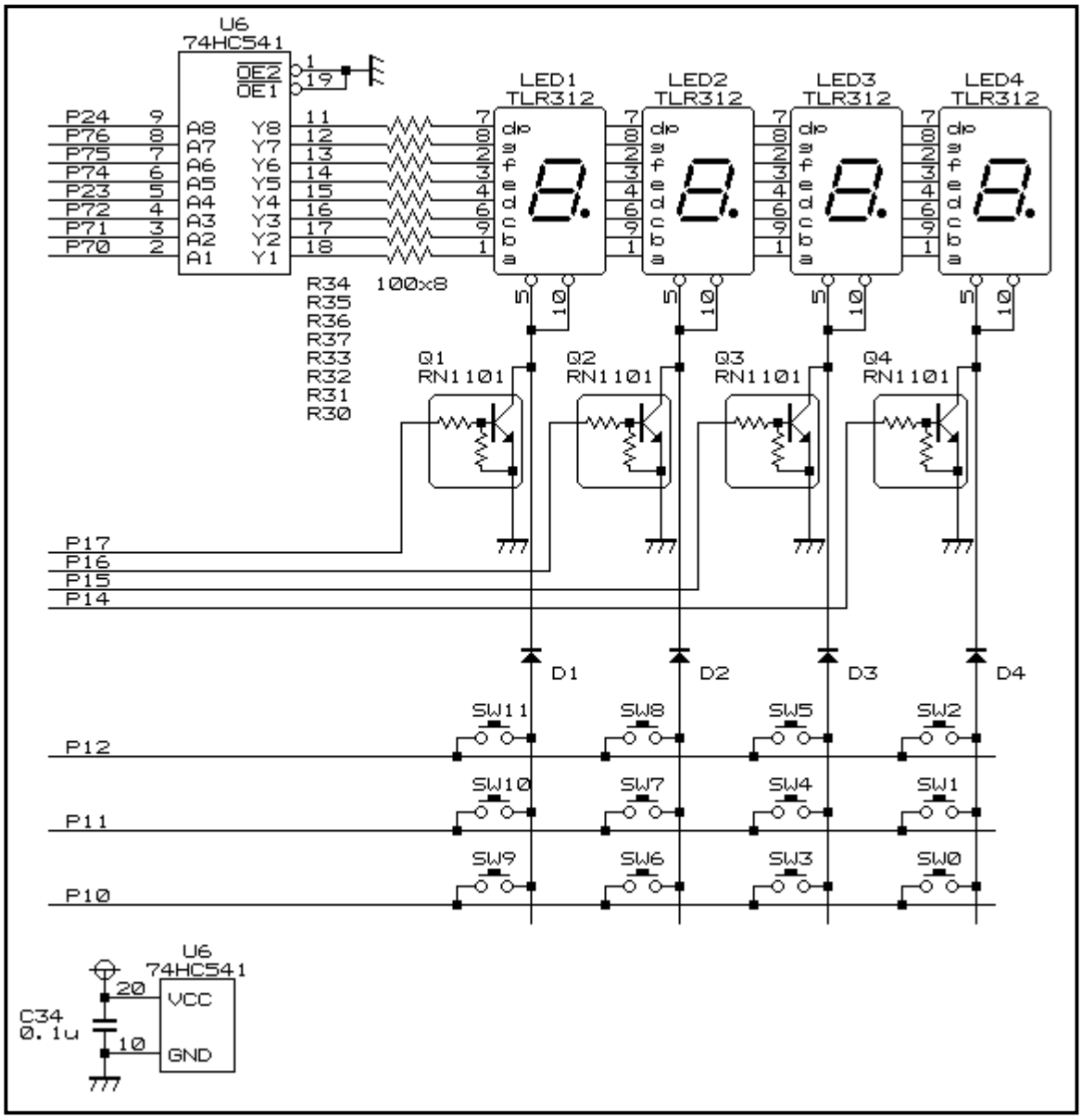
```
                }

                // 押されたキー番号を得る，キー番号が若い方が優先
                if (equal==OK){         // 1st=2ndのとき
                    for (i=KETA-1;i>=0;i--){
                        KeyBuf[i] = Key2ndBuf[i];
                        if      ((KeyBuf[i]&0x01)==0x01){
                            key = i*3+0; push = 1;
                        }
                        else if ((KeyBuf[i]&0x02)==0x02){
                            key = i*3+1; push = 1;
                        }
                        else if ((KeyBuf[i]&0x04)==0x04){
                            key = i*3+2; push = 1;
                        }
                    }

                    if (push==1){ // 押されている
                        if (KeyNoOld!=key){// 前回と違うキーが押されている
                            KeyNo = key; KeyNoOld = key;
                            KeyFlag = 1;
                        }
                    }
                    else{               // 何も押されていない
                        KeyNoOld = -1;
                    }
                }
            }
            break;
    }
}


/*************************************************************************
    LED スキャン
*************************************************************************/
void scan(void)
{
    //表示消去
    P_PORT.PDR7.BYTE = 0x00;
    P_PORT.PDR2.BYTE = P_PORT.PDR2.BYTE & 0xe7;

    //スキャンラインオールディセーブル
    P_PORT.PDR1.BYTE = P_PORT.PDR1.BYTE & 0x0f;

    //スキャンカウンタの更新
    ScanPnt++; if (ScanPnt>(KETA-1)) ScanPnt=0;

    //次のセグメントを表示する
    P_PORT.PDR7.BYTE = SegBuf[ScanPnt] & 0x77;
    P_PORT.PDR2.BIT.P23 = ((SegBuf[ScanPnt]&0x08) ? 1 : 0);
    P_PORT.PDR2.BIT.P24 = ((SegBuf[ScanPnt]&0x80) ? 1 : 0);
    switch (ScanPnt){
        case 0:
            P_PORT.PDR1.BIT.P14 = 1; break;
        case 1:
            P_PORT.PDR1.BIT.P15 = 1; break;
        case 2:
```

```
            P_PORT.PDR1.BIT.P16 = 1; break;
        case 3:
            P_PORT.PDR1.BIT.P17 = 1; break;
    }
}


/************************************************************************
    タイマB1 イニシャライズ
************************************************************************/
void init_tmrb1(void)
{
    P_TMRB1.TMB1.BYTE  =   0xfb;         //オートリロード,X'tal/256
    P_TMRB1.TCB1       =   256-195;  //2.496ms
}


/************************************************************************
    IOポート イニシャライズ
************************************************************************/
void init_port(void)
{
    P_PORT.PMR1.BYTE   =   0x00;     //P10,11,14-17,22,72 使用
    P_PORT.PCR1.BYTE   =   0xf0;     //P10-12 In / P14-17 Out
    P_PORT.PUCR1.BYTE  =   0x00;     //プルアップMOS使用しない
    P_PORT.PDR1.BYTE   =   0x00;     //P14-17 = Low

    P_PORT.PCR2.BYTE   =   0x18;     //P20-22 In / P23,24 Out
    P_PORT.PDR2.BYTE   =   0x00;     //P23,24 = Low
    P_PORT.PMR3.BYTE   =   0x00;     //P23,24 CMOS Out

    P_PORT.PCR7.BYTE   =   0x77;     //P70-72,74-76 Out
    P_PORT.PDR7.BYTE   =   0x00;     //P70-72,74-76 = Low
}


/************************************************************************
    割込みコントローラ イニシャライズ
************************************************************************/
void init_int(void)
{
    P_INT.IRR2.BIT.IRRTB1   =   0;   //タイマB1割込み要求フラグクリア
    P_INT.IENR2.BIT.IENTB1  =   1;   //タイマB1割込み要求イネーブル
}


/************************************************************************
    End of Dynamic Scan Program.
************************************************************************/
```

このプログラムは割り込みを使っているので HEW が自動生成する‘intprg. c’を修正する必要があります。修正後のリストは次の通りです。

```
/************************************************************************/
/*                                                                      */
/*  FILE        :intprg.c                                               */
/*  DATE        :Tue, Jan 20, 2004                                      */
/*  DESCRIPTION :Interrupt Program                                      */
/*  CPU TYPE    :H8/3687                                                */
/*                                                                      */
/*  This file is generated by Hitachi Project Generator (Ver.2.1).      */
/*                                                                      */
/************************************************************************/

#include <machine.h>

extern void intprog_tmrb1(void);        //←追加 ------------------------

#pragma section IntPRG
//  vector 1 Reserved

//  vector 2 Reserved

//  vector 3 Reserved
        ～



//  vector 22 Timer V
__interrupt(vect=22) void INT_TimerV(void) {/* sleep(); */}
//  vector 23 SCI3
__interrupt(vect=23) void INT_SCI3(void) {/* sleep(); */}
//  vector 24 IIC2
__interrupt(vect=24) void INT_IIC2(void) {/* sleep(); */}
//  vector 25 ADI
__interrupt(vect=25) void INT_ADI(void) {/* sleep(); */}
//  vector 26 Timer Z0
__interrupt(vect=26) void INT_TimerZ0(void) {/* sleep(); */}
//  vector 27 Timer Z1
__interrupt(vect=27) void INT_TimerZ1(void) {/* sleep(); */}
//  vector 28 Reserved

//  vector 29 Timer B1
__interrupt(vect=29) void INT_TimerB1(void){intprog_tmrb1();}    //←修正 --------------
//  vector 30 Reserved

//  vector 31 Reserved

//  vector 32 SCI3_2
__interrupt(vect=32) void INT_SCI3_2(void) {/* sleep(); */}
```

4. 表示&キーの全体回路図



第2章 AD コンバータ

自然界の物理量、例えば、温度、湿度、重さ、音声等は全てアナログ量です。一方、マイコンはデジタル値しか取り扱うことができません。ということは、マイコンで物理量を取り扱うためにはアナログ値を何らかの方法でデジタル値に変換する必要があります。このような働きをするデバイスが A/D コンバータです。

H8/3687 の A/D コンバータは 10bit の分解能を持ってます。これはどれくらいの分解能かといいますと 0～5V の電圧を 1024 ステップに分解することができます。つまり 1 ステップあたりの電圧変化量は 5V を 1024 ステップで割るので約 0.005V(5mV)となります。また、変換時間も 1 チャンネルあたり最小 3.5μsec と高機能な A/D コンバータです。

この章では H8/3687 の A/D コンバータの基本的な使い方と平均化の方法を学習します。まず A/D コンバータで使用するレジスタの説明から始まり、1A/D 変換プログラム、平均化処理の仕方、そして平均化処理を加えた A/D 変換プログラムと基礎を学び、応用としてスキャンモードを使用した 4 チャンネル同時変換プログラムの説明をします。

1. レジスタ説明

まず A/D 変換で使用するレジスタの説明をします。A/D 変換には次のレジスタがあります。

■ A/D データレジスタ A～D （ADDRA～D）

A/D 変換結果を格納するための 16bit リード専用レジスタで、A～D まで 4 つあります。10bit の変換データは bit15 から bit6 に格納され下位 6bit の値は常に 0 です。各アナログ入力チャンネルの変換結果が格納される A/D データレジスタは以下の通りです。

アナログ入力チャンネル		変換結果が格納される A/D データレジスタ
グループ 0	グループ 1	
AN0	AN4	ADDRA
AN1	AN5	ADDRB
AN2	AN6	ADDRC
AN3	AN7	ADDRD

■ A/D コントロール／ステータスレジスタ （ADCSR）

ADCSR は A/D 変換器の制御ビットと変換終了ステータスビットで構成されている 8bit レジスタです。

ビット	7	6	5	4	3	2	1	0
ビット名	ADF	ADIE	ADST	SCAN	CKS	CH2	CH1	CH0

ビット	ビット名	説　明
7	ADF	A/D エンドフラグ 単一モードでは A/D 変換が終了した時、スキャンモードでは選択した全てのチャンネルの A/D 変換が終了した時、セットされます
6	ADIE	A/D インタラプトイネーブル このビットを 1 にすると ADF による変換終了割り込み要求がイネーブルになります
5	ADST	A/D スタート このビットを 1 にセットすると A/D 変換を開始します 単一モードでは変換が終了すると自動的にクリアされます スキャンモードではクリアされるまで選択されたチャンネルを順次連続変換します

4	SCAN	スキャンモード A/D 変換のモードを選択します 0:単一モード 1:スキャンモード
3	CKS	クロックセレクト A/D 変換時間の設定を行います 0:変換時間＝134 ステート(max) 1:変換時間＝70 ステート(max) 変換の切り替えは ADST＝0 の状態で行って下さい
2 1 0	CH2 CH1 CH0	チャンネルセレクト 2～0 アナログ入力チャンネルを選択します SCAN=0　　SCAN=1 000:AN0　　000:AN0 001:AN1　　001:AN0～AN1 010:AN2　　010:AN0～AN2 011:AN3　　011:AN0～AN3 100:AN4　　100:AN4 101:AN5　　101:AN4～AN5 110:AN6　　110:AN4～AN6 111:AN7　　111:AN4～AN7

■ **A/D コントロールレジスタ （ADCR）**

ADCR は外部トリガによる A/D 変換開始をイネーブルにします

ビット	ビット名	説　明
7	TRGE	トリガイネーブル このビットを 1 にすると外部トリガ端子(ADTRG)の立ち上がり、立下りでも A/D 変換を開始します
6～1	－	リザーブビットです
0	－	リザーブビットです ※1 に設定しないで下さい

※このドキュメント内で作成するプログラムは外部トリガを使用しませんので、このレジスタは使用しません。

2. 1 変換プログラム

まずは基礎となる 1 変換のプログラムを行います。この後、平均化処理やスキャンモードでの複数チャンネル変換を行いますが、どれも基本的な変換方法は同じですので、ここでしっかりとマスターしましょう。

1 チャンネルのみ変換を行い、その結果を 7 セグメント LED に表示します。1 チャンネルのみの変換ですの A/D 変換モードは単一モードです。変換結果を値として表示するので入力信号は直流的な信号と想定します。直流信号なら変換スピードは要求されませんので、変換時間は 134 ステートとします。尚、アナログ入力チャンネルは AN0 とします。

1 変換ですので更新間隔＝変換間隔となります。直流信号なので極端に早く更新する必要はありません。かといって 1 秒程度では少しゆっくりな感じがします。そこで更新間隔を 1 秒間に 4 回程度とします。タイマ Z で 1msec 毎に割り込みを掛け、256 カウントしたら ADCSR の ADST を 1 にセットして変換を開始させます。つまり更新間隔は 256msec 毎です。

変換が終了すると ADCSR の ADF がセットされます。変換終了を検出するには、メインループ等で ADF がセットされるのを待つ方法と割り込みで検出する方法の 2 種類あります。ここでは割り込みによって変換終了を検出します。ADCSR の ADIE ビットを 1 にセットすると A/D 変換終了で割り込みがかかるようになります。

以上から A/D コントロール／ステータスレジスタ（ADCSR）にセットする値は次のようになります。

7	6	5	4	3	2	1	0	ADCSR
ADF	ADIE	ADST	SCAN	CKS	CH2	CH1	CH0	設定値
0	1	0	0	0	0	0	0	0x40

後はタイマ Z の割り込みで ADST=1 とすれば 1 変換を開始し、変換終了で A/D 変換終了割り込みがかかります。尚、A/D 変換、タイマ Z 共にそれぞれの割り込みイネーブルビットをセットしコンディションレジスタ（CCR）の I ビットをクリアすれば割り込みが発生します。

変換結果は A/D データレジスタ A（ADDRA）にセットされていますが、レジスタの説明で記したように変換結果は bit15～bit6 にセットされていて下 6 ビット（bit5～bit0）は全て 0 です。このままでは扱いにくいので右に 6 ビットシフトして右詰めにします。そしてこの右詰されたデータをセグメント変換し表示します。

次ページに 1 変換プログラムのリストを記します。タイマ Z の設定はリストを参照して下さい。又、7 セグメント LED のスキャンプログラムはダイナミックスキャンからの流用ですのでここでは説明を省きます。

```
/*********************************************************************/
/*                                                                   */
/*  TK-3687 I/F トレーニングユニット                                 */
/*  No,2 AD コンバータ                                               */
/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */
/*  FILE        :adc_one.c                                           */
/*  DATE        :Thu, Apr 22, 2004                                   */
/*  DESCRIPTION :Main Program                                        */
/*  CPU TYPE    :H8/3687                                             */
/*                                                                   */
/*  This file is generated by Hitachi Project Generator (Ver.2.1).   */
/*               programed by Toyo-linx,Co.,Ltd. / Y.Furukawa.       */
/*                                                                   */
/*********************************************************************/
/*
■概要
  単一モードで AN0 を A/D 変換し 7segLED に表示します。変換タイミングは 256msec
毎で、このタイミングはタイマＺの 1msec 毎の割り込みを 256 カウントして作って
います。タイマＺ1msec×256 カウントで変換開始。 A/D の変換終了割り込みで値
を表示します。
*/

/*********************************************************************
    履歴
*********************************************************************/
/*
2004-04-22    作成開始
*/


/*********************************************************************
    インクルードファイル
*********************************************************************/
#include  <machine.h>
#include  "iodefine.h"


/*********************************************************************
    定数の定義（直接指定）
*********************************************************************/
// A/D に関する定数 ----------------------------------------------------
#define       AVE_CNST      256       // 平均化回数

// 7 セグメント LED に関係する定数 -------------------------------------
#define       KETA          4         //7 セグメント LED の桁数

// その他 --------------------------------------------------------------
#define       OK                0         //戻り値
#define       NG                -1        //戻り値


/*********************************************************************
    定数の定義（ROM）
*********************************************************************/
// 7 セグメントデータ変換テーブル --------------------------------------
```

```
const unsigned charSegTable[16] = {0x3f,0x06,0x5b,0x4f,   //0,1,2,3
                                   0x66,0x6d,0x7d,0x07,   //4,5,6,7
                                      0x7f,0x67,0x77,0x7c,    //8,9,A,B
                                      0x39,0x5e,0x79,0x71};   //C,D,E,F


/************************************************************************
    スタティック変数の定義とイニシャライズ
    ここでイニシャライズしていない変数はプログラム中で行なっている
************************************************************************/
// 7 セグメント LED スキャンに関する変数 ---------------------------------
unsigned char  SegBuf[4] = {0x3f   //7 セグメント LED バッファ
                           ,0x3f   //（初期表示）
                              ,0x3f
                              ,0xbf};
unsigned char  ScanPnt = 0;       //どのスキャンラインか
                                  //（キースキャンと兼用）

// キースキャンに関する変数 ----------------------------------------------
unsigned char  Key1stBuf[4];      //キー1st リード
unsigned char  Key2ndBuf[4];      //キー2nd リード
unsigned char  KeyBuf[4];             //1st=2nd になったキーデータ
unsigned char  KeyReadFlag = 0;   //キーリードフラグ
                                  // 0:1st リード
                                  // 1:2nd リード
int               KeyNo = 0;              //キー番号
int               KeyNoOld = -1;      //前回のキー番号
unsigned char  KeyFlag = 0;       //キー入力フラグ
                                  // 1:キー入力あり

// AD コンバータに関する変数 ---------------------------------------------
unsigned int   TZcount = 0;   // ﾀｲﾏ Z ｶｳﾝﾄ(変換ﾀｲﾐﾝｸﾞ)
unsigned char  ADFlag = 0;        // 変換終了ﾌﾗｸﾞ
unsigned long  ADBuf = 0;         // A/D ﾊﾞｯﾌｧ


/************************************************************************
    関数の定義
************************************************************************/
void          init_int(void);
void          init_port(void);
void          init_tmrz(void);
void          init_tmrb1(void);
void          init_adc(void);

void          main(void);
void          scan(void);
void          disp(void);
void          keyin(void);

void          eoc_ave(void);      // A/D 変換終了割り込み / 加算と平均化
void          ad_start(void);         // タイマ Z 割り込み / A/D 変換開始
void          set_disp(unsigned int); // 表示データセット
void          chk_key(void);      // キー入力チェック


/************************************************************************
    メインプログラム
```

```
*************************************************************************/
void main(void)
{
//----- イニシャライズ ----------------------------------------------
     init_tmrz();
     init_tmrb1();
     init_port();
     init_int();
     init_adc();

     P_TMRZ.TSTR.BIT.STR0 = 1;    // ﾀｲﾏ Z TCNT_0 ｶｳﾝﾄ開始

     set_imask_ccr(0);  //割り込みイネーブル

//----- メインループ ------------------------------------------------
     while(1){
          if (ADFlag == 1)   set_disp(ADBuf);   // 表示ﾃﾞｰﾀｾｯﾄ
     }
//-------------------------------------------------------------------
}


/*************************************************************************
     Ａ／Ｄ変換開始（ﾀｲﾏ Z 割り込み）
*************************************************************************/
#pragma regsave (ad_start)
void ad_start(void)
{
     P_TMRZ0.TSR0.BIT.IMFA = 0;        // ｺﾝﾍﾟｱﾏｯﾁﾌﾗｸﾞ A ｸﾘｱ
     TZcount++;                             // ｶｳﾝﾄ
     if (TZcount >= 256){
          TZcount = 0;                 // ｶｳﾝﾄ ｸﾘｱ
          P_AD.ADCSR.BIT.ADST = 1;     // A/D 変換 ｽﾀｰﾄ
     }
}


/*************************************************************************
     Ａ／Ｄ平均化処理（A/D 変換終了割り込み）
*************************************************************************/
#pragma regsave (eoc_ave)
void eoc_ave(void)
{
     int  i;


     P_AD.ADCSR.BIT.ADST=    0;   // ADCSR:ADST ｸﾘｱ
     P_AD.ADCSR.BIT.ADF =    0;   // ADCSR:ADF ｸﾘｱ

     ADBuf = (P_AD.DRA>>6);       // AN0 A/D 値ﾘｰﾄﾞ
     ADFlag = 1;                       // 変換終了ﾌﾗｸﾞ ｾｯﾄ
}


/*************************************************************************
     キー&LED スキャン（タイマ B1 割り込み）
*************************************************************************/
#pragma regsave (scan)
```

```
void scan(void)
{
      ｛
      ｝
}


/*************************************************************************
    キー スキャン
*************************************************************************/
void keyin(void)
{
      ｛
      ｝
}


/*************************************************************************
    LED スキャン
*************************************************************************/
void disp(void)
{
      ｛
      ｝
}


/*************************************************************************
    表示データセット
*************************************************************************/
void set_disp(unsigned int data)
{
    set_imask_ccr(1);  // 割り込みﾃﾞｨｾｰﾌﾞﾙ

    ADFlag = 0;             // 平均化終了ﾌﾗｸﾞ ｸﾘｱ

    SegBuf[3] = 0;      // A/Dch ｾｯﾄ
    SegBuf[2] = SegTable[(data>>8)&0x03]; // A/D 値(HEX) 3 桁目 ｾｯﾄ
    SegBuf[1] = SegTable[(data>>4)&0x0f]; // A/D 値(HEX) 2 桁目 ｾｯﾄ
    SegBuf[0] = SegTable[data&0x0f];      // A/D 値(HEX) 1 桁目 ｾｯﾄ

    set_imask_ccr(0);  // 割り込みｲﾈｰﾌﾞﾙ
}


/*************************************************************************
    タイマZ イニシャライズ
*************************************************************************/
void init_tmrz(void)
{
    P_TMRZ0.TCR0.BYTE  =    0x23;     // CLK=X'tal/8,GRA ｺﾝﾍﾟｱﾏｯﾁ TCNT ｸﾘｱ
    P_TMRZ0.TIORA0.BYTE=    0x88;     // GRA ｱｳﾄﾌﾟｯﾄｺﾝﾍﾟｱﾚｼﾞｽﾀ,端子出力禁止
    P_TMRZ0.GRA0       =    2500;     // 周期設定 / 1msec
    P_TMRZ0.TIER0.BIT.IMIEA = 1; // TCNT=GRA ｲﾝﾀﾗﾌﾟﾄ ｲﾈｰﾌﾞﾙ
}
```

```
/*************************************************************************
    タイマＢ１ イニシャライズ
*************************************************************************/
void init_tmrb1(void)
{
    P_TMRB1.TMB1.BYTE  =   0xfb;         // オートリロード,X'tal/256
    P_TMRB1.TCB1       =   256-195;  // 2.496msec
}


/*************************************************************************
    Ｉ／Ｏポート イニシャライズ
*************************************************************************/
void init_port(void)
{
    P_PORT.PMR1.BYTE   =   0x00;     //P10,11,14-17,22,72 使用
    P_PORT.PCR1.BYTE   =   0xf0;     //P10-12 In / P14-17 Out
    P_PORT.PUCR1.BYTE  =   0x00;     //プルアップ MOS 使用しない
    P_PORT.PDR1.BYTE   =   0x00;     //P14-17 = Low

    P_PORT.PCR2.BYTE   =   0x18;     //P20-22 In / P23,24 Out
    P_PORT.PDR2.BYTE   =   0x00;     //P23,24 = Low
    P_PORT.PMR3.BYTE   =   0x00;     //P23,24 CMOS Out

    P_PORT.PCR6.BYTE   =   0xFF;     //P60-67 Out
    P_PORT.PDR7.BYTE   =   0x00;     //P60-67 = Low

    P_PORT.PCR7.BYTE   =   0x77;     //P70-72,74-76 Out
    P_PORT.PDR7.BYTE   =   0x00;     //P70-72,74-76 = Low
}


/*************************************************************************
    割込みコントローラ イニシャライズ
*************************************************************************/
void init_int(void)
{
    P_INT.IRR2.BIT.IRRTB1   =   0;   //タイマ B1 割込み要求フラグクリア
    P_INT.IENR2.BIT.IENTB1  =   1;   //タイマ B1 割込み要求イネーブル
}


/*************************************************************************
    Ａ／Ｄ変換器 イニシャライズ
*************************************************************************/
void init_adc(void)
{
    P_AD.ADCSR.BYTE    =   0x40;     // 単一ﾓｰﾄﾞ AN0,134 ｽﾃｰﾄ,ｲﾝﾀﾗﾌﾟﾄｲﾈｰﾌﾞﾙ
}


/*************************************************************************
    End of Dynamic Scan Program.
*************************************************************************/
```

```
/************************************************************************/
/*                                                                      */
/*  FILE        :intprg.c                                               */
/*  DATE        :Thu, Apr 22, 2004                                      */
/*  DESCRIPTION :Interrupt Program                                      */
/*  CPU TYPE    :H8/3687                                                */
/*                                                                      */
/*  This file is generated by Hitachi Project Generator (Ver.2.1).      */
/*                                                                      */
/************************************************************************/

#include <machine.h>
#include                        "iodefine.h"
#pragma section IntPRG
      (
      )
//  vector 25 ADI
__interrupt(vect=25) void INT_ADI(void)
{
                            P_PORT.PDR6.BIT.P67 = 1;// P67 変換終了処理中 High ﾊﾟﾙｽ
                                   eoc_ave();
                            P_PORT.PDR6.BIT.P67 = 0;
}

//  vector 26 Timer Z0
__interrupt(vect=26) void INT_TimerZ0(void)
{
                            P_PORT.PDR6.BIT.P66 = 1;// P66 ﾀｲﾏ Z0 処理時間中 High ﾊﾟﾙｽ
                                  ad_start();
                            P_PORT.PDR6.BIT.P66 = 0;
}

//  vector 27 Timer Z1
__interrupt(vect=27) void INT_TimerZ1(void) {/* sleep(); */}
//  vector 28 Reserved

//  vector 29 Timer B1
__interrupt(vect=29) void INT_TimerB1(void)
{
                            P_PORT.PDR6.BIT.P65 = 1;// P65 ﾀｲﾏ B1 処理時間中 High ﾊﾟﾙｽ
                                    scan();
                            P_PORT.PDR6.BIT.P65 = 0;
}

//  vector 30 Reserved

//  vector 31 Reserved

//  vector 32 SCI3_2
__interrupt(vect=32) void INT_SCI3_2(void) {/* sleep(); */}
```

プログラムを実行したら、CN1 にボリュームを繋いで入力を変化させて見ましょう。変化することが確認できればA/D 変換は正常に行われています。それではボリュームを適当なところで固定してみましょう。値がふらついているのが確認できると思います。一体何故でしょうか？

3. 平均化処理

入力されている信号源は必ずしも安定しているとは限りませんし、絶対的に安定している信号などありません。その為 A/D 変換結果もまた安定していません。前章でボリュームを適当なところで固定した時に表示がふらついていたのはこのためです。そこで何回か連続して変換を行いその平均値を採用します。

平均化処理を追加した A/D 変換部のリストを以下に記します。このプログラムでは平均化回数を定数 AVE_CNST で定義し、AVE_CNST 回変換・加算を行い、加算結果を右詰め（6 ビット右シフト）してから AVE_CNST で割り平均値を求めています。ここでは AVE_CNST＝256 としました。前章では変換間隔としてタイマ Z の 1msec 割り込みを 256 回カウントしていましたが、ここでは加算回数が変換間隔の役割も果たしますので、1msec 毎に変換を行います。A/D 値の加算は ADBuf で行い、平均化した値は ADAveBuf にセットします。表示はこの ADAveBuf の値をしようします。尚、前回変換終了を表していたフラグ ADFlag は、ここでは平均化終了を表す為 ADAveFlag に名称を変更しています。

```
/************************************************************************
     Ａ／Ｄ変換開始（ﾀｲﾏ Z 割り込み）
************************************************************************/
#pragma regsave (ad_start)
void ad_start(void)
{
     P_TMRZ0.TSR0.BIT.IMFA = 0;   // ｺﾝﾍﾟｱﾏｯﾁﾌﾗｸﾞ A ｸﾘｱ
     P_AD.ADCSR.BIT.ADST = 1;     // A/D 変換 ｽﾀｰﾄ
}


/************************************************************************
     Ａ／Ｄ平均化処理（A/D 変換終了割り込み）
************************************************************************/
#pragma regsave (eoc_ave)
void eoc_ave(void)
{
     int  i;


     P_AD.ADCSR.BIT.ADST=    0;   // ADCSR:ADST ｸﾘｱ
     P_AD.ADCSR.BIT.ADF =    0;   // ADCSR:ADF ｸﾘｱ

     ADBuf += P_AD.DRA; // AN0 A/D 値ﾘｰﾄﾞ&加算

     if (++ADAveCnt==AVE_CNST){   // 加算回数+1 加算終了?
          // 加算終了・平均化
          ADAveData = (ADBuf>>6)/AVE_CNST;  // 平均化

          ADAveFlag = 1; // 平均化ﾌﾗｸﾞ ｾｯﾄ

          ADAveCnt = 0;  // 加算回数 ｸﾘｱ
          ADBuf = 0;          // 加算ﾊﾞｯﾌｧ ｸﾘｱ
     }
}
```

4. スキャンモードでの 4 チャンネル同時変換

前章で基本的な A/D 変換はマスタできたことと思います。この章では A/D 変換器のスキャンモードを使用して 4 チャンネル同時変換を行います。とは言っても変換・平均化処理は前章と何ら変わりません。ただ今まで 1 チャンネルだったのが 4 チャンネルになるだけのことです。

スキャンモードはどのようなモードなのでしょうか？H8/3687 ハードウェアマニュアルに記載されている説明を以下に記します。

スキャンモード
スキャンモードは指定された最大4 チャネルのアナログ入力を以下のように順次連続してA/D 変換します。

1. ソフトウェアまたは外部トリガ入力によってADCSRのADSTビットが1にセットされると、グループの第1チャネル（CH2＝0のときAN0、CH2＝1のときAN4）からA/D変換を開始します。
2. それぞれのチャネルのA/D変換が終了するとA/D変換結果は順次そのチャネルに対応するA/Dデータレジスタに転送されます。
3. 選択されたすべてのチャネルのA/D変換が終了するとADCSRのADFフラグが1にセットされます。このとき、ADIEビットが1にセットされていると、ADI割り込み要求を発生します。A/D変換器は再びグループの第1チャネルからA/D変換を開始します。

ADSTビットは自動的にはクリアされず、1 にセットされている間は、2.～3.を繰り返します。ADSTビットを 0 にクリアすると A/D 変換は停止します。

つまり、今まで通り A/D 変換終了割り込みで選択した各チャンネルの A/D 値を読み出し、加算・平均化すればよいのです。しかしここで注意しなくてはならないことがひとつあります。それは ADST ビットをクリアしない限り変換を繰り返すということです。これは変換終了割り込みを掛けている場合、ADST ビットをクリアしないと 1 変換時間×選択チャンネル数毎に割り込みが発生することを示しています。ここで作成するプログラムは前回同様タイマ Z の 1msec 割り込みで 1 変換を行いたいので、変換終了割り込みを受け付けたら ADST ビットをクリアして A/D 変換を停止させます。

I/F トレーニングユニットのアナログチャンネルは AN0～AN3 まで接続されているので 4 チャンネル指定します。AN0～AN3、スキャンモード時の ADSCR の設定値を以下に示します。

7	6	5	4	3	2	1	0	ADCSR
ADF	ADIE	ADST	SCAN	CKS	CH2	CH1	CH0	設定値
0	1	0	1	0	0	1	1	0x53

表示チャンネルの切り替えはスイッチに各チャンネルを割り付けます。割り付けられたスイッチが押されたら 7 セグメント LED の最左桁にチャンネル番号を表示し、残り 3 桁で A/D 値を表示します。チャンネル番号と A/D 値を区別する為にチャンネル番号の桁には d.p.を点灯させます。

以上を踏まえたプログラムのリストを次ページに示します。

```
/************************************************************************/
/*                                                                      */
/*  TK-3687 I/F トレーニングユニット                                    */
/*  No,2 AD コンバータ                                                  */
/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */
/*  FILE        :adc.c                                                  */
/*  DATE        :Tue, Feb 03, 2004                                      */
/*  DESCRIPTION :Main Program                                           */
/*  CPU TYPE    :H8/3687                                                */
/*  BOARD       :TK-3687(B6081) & I/F-Unit(B6084)                       */
/*                                                                      */
/*  This file is generated by Hitachi Project Generator (Ver.2.1).      */
/*               programed by Toyo-linx,Co.,Ltd. / Y.Furukawa.          */
/*                                                                      */
/************************************************************************/
/*
■概要
  スキャンモードで AN0～AN3 を A/D 変換し、選択されているチャンネルの A/D 値を
7segLED に表示します。変換タイミングは 1msec 毎で、このタイミングはタイマＺ
で作っています。タイマＺの割り込みで変換開始。 A/D の変換終了割り込みで平
均化処理を行います。平均化加算回数は“ AVE_CNST ”で定義された回数です。
表示は上桁にチャンネル(コネクタ)番号、残り３桁で A/D 値を１６進数で表示しま
す。尚、チャンネル番号と A/D 値を区別する為にチャンネル番号には d.p.を点灯し
てます。
チャンネルの切り替えは基板上のキー１～４で選択します。
*/

/************************************************************************
    履歴
************************************************************************/
/*
2004-02-03    作成開始
    02-13     adc_one 追加
     02-25    ﾀｲﾏ Z で A/D 変換開始
              変換終了割り込みで平均化処理
     02-26    A/D 変換をｽｷｬﾝﾓｰﾄﾞに変更
     03-02    I/F ﾕﾆｯﾄ接続の為の修正
     04-02    小修正 ｺﾒﾝﾄ追加
    04-07     resetprg.c にﾊｲﾊﾟｰ H8 組み込み用の nop();x4 追加
*/


/************************************************************************
    インクルードファイル
************************************************************************/
#include  <machine.h>
#include  "iodefine.h"


/************************************************************************
    定数の定義（直接指定）
************************************************************************/
// A/D に関する定数 ----------------------------------------------------
#define       AVE_CNST      256       // 平均化回数

// 7 セグメント LED に関係する定数 -------------------------------------
```

```
#define       KETA          4         //7 セグメント LED の桁数

// その他 ----------------------------------------------------------
#define       OK                0         //戻り値
#define       NG                -1        //戻り値


/*********************************************************************
    定数の定義（ROM）
*********************************************************************/
// 7 セグメントデータ変換テーブル ----------------------------------------
const unsigned charSegTable[16] = {0x3f,0x06,0x5b,0x4f,   //0,1,2,3
                                   0x66,0x6d,0x7d,0x07,   //4,5,6,7
                                      0x7f,0x67,0x77,0x7c,    //8,9,A,B
                                      0x39,0x5e,0x79,0x71};   //C,D,E,F


/*********************************************************************
    スタティック変数の定義とイニシャライズ
    ここでイニシャライズしていない変数はプログラム中で行なっている
*********************************************************************/
// 7 セグメント LED スキャンに関する変数 ---------------------------------
unsigned char  SegBuf[4] = {0x3f  //7 セグメント LED バッファ
                           ,0x3f  //（初期表示）
                             ,0x3f
                             ,0xbf};
unsigned char  ScanPnt = 0;       //どのスキャンラインか
                                        //（キースキャンと兼用）

// キースキャンに関する変数 ----------------------------------------------
unsigned char  Key1stBuf[4];      //キー1st リード
unsigned char  Key2ndBuf[4];      //キー2nd リード
unsigned char  KeyBuf[4];             //1st=2nd になったキーデータ
unsigned char  KeyReadFlag = 0;   //キーリードフラグ
                                        // 0:1st リード
                                        // 1:2nd リード
int                KeyNo = 0;             //キー番号
int                KeyNoOld = -1;     //前回のキー番号
unsigned char  KeyFlag = 0;       //キー入力フラグ
                                        // 1:キー入力あり

// AD コンバータに関する変数 ---------------------------------------------
int                ADch = 0;          // 変換ﾁｬﾝﾈﾙ No,
unsigned char  ADAveFlag = 0;     // 平均化終了ﾌﾗｸﾞ
unsigned int   ADAveCnt = 0;      // 平均化加算ｶｳﾝﾄﾊﾞｯﾌｧ
unsigned long  ADBuf[4]           // A/D 加算ﾊﾞｯﾌｧ
                       ={0,0,0,0};
unsigned int   ADAveData[4]       // 平均化 A/D 値
                       ={0,0,0,0};

/*********************************************************************
    関数の定義
*********************************************************************/
void           init_int(void);
void           init_port(void);
void           init_tmrz(void);
void           init_tmrb1(void);
void           init_adc(void);
```

```
void          main(void);
void          scan(void);
void          disp(void);
void          keyin(void);

void          eoc_ave(void);      // A/D 変換終了割り込み / 加算と平均化
void          ad_start(void);          // タイマＺ割り込み / A/D 変換開始
void          set_disp(unsigned int); // 表示データセット
void          chk_key(void);      // キー入力チェック


/************************************************************************
     メインプログラム
************************************************************************/
void main(void)
{
//----- イニシャライズ ---------------------------------------------
     init_tmrz();
     init_tmrb1();
     init_port();
     init_int();
     init_adc();

     P_TMRZ.TSTR.BIT.STR0 = 1;    // ﾀｲﾏ Z TCNT_0 ｶｳﾝﾄ開始

     set_imask_ccr(0);  //割り込みイネーブル

//----- メインループ -----------------------------------------------
     while(1){
          if (ADAveFlag == 1)set_disp(ADAveData[ADch]);   // 表示データセット
          if (KeyFlag == 1)  chk_key();     // キー入力チェック
     }
//------------------------------------------------------------------
}


/************************************************************************
     Ａ／Ｄ変換開始（ﾀｲﾏ Ｚ割り込み）
************************************************************************/
#pragma regsave (ad_start)
void ad_start(void)
{
     P_TMRZ0.TSR0.BIT.IMFA = 0;   // ｺﾝﾍﾟｱﾏｯﾁﾌﾗｸﾞ A ｸﾘｱ
     P_AD.ADCSR.BIT.ADST = 1;     // A/D 変換 ｽﾀｰﾄ
}


/************************************************************************
     Ａ／Ｄ平均化処理（A/D 変換終了割り込み）
************************************************************************/
#pragma regsave (eoc_ave)
void eoc_ave(void)
{
     int  i;


     P_AD.ADCSR.BIT.ADST=    0;   // ADCSR:ADST ｸﾘｱ
```

```
    P_AD.ADCSR.BIT.ADF =    0;   // ADCSR:ADF ｸﾘｱ

    ADBuf[0] += P_AD.DRA;   // AN0 A/D 値ﾘｰﾄﾞ&加算
    ADBuf[1] += P_AD.DRB;   // AN1 A/D 値ﾘｰﾄﾞ&加算
    ADBuf[2] += P_AD.DRC;   // AN2 A/D 値ﾘｰﾄﾞ&加算
    ADBuf[3] += P_AD.DRD;   // AN3 A/D 値ﾘｰﾄﾞ&加算

    if (++ADAveCnt==AVE_CNST){   // 加算回数+1 加算終了?
        // 加算終了・平均化
        for (i=0;i<4;i++)
            ADAveData[i] = (ADBuf[i]>>6)/AVE_CNST; // 平均化

        ADAveFlag = 1; // 平均化ﾌﾗｸﾞ ｾｯﾄ

        ADAveCnt = 0; // 加算回数 ｸﾘｱ
        for (i=0;i<4;i++)
            ADBuf[i] = 0; // 加算ﾊﾞｯﾌｧ ｸﾘｱ
    }
}


/************************************************************************
    キー&LED スキャン（タイマ B1 割り込み）
************************************************************************/
#pragma regsave (scan)
void scan(void)
{
        (
        )
}


/************************************************************************
    キー スキャン
************************************************************************/
void keyin(void)
{
        (
        )
}


/************************************************************************
    LED スキャン
************************************************************************/
void disp(void)
{
        (
        )
}


/************************************************************************
    表示データセット
************************************************************************/
```

```
void set_disp(unsigned int data)
{
    set_imask_ccr(1);  // 割り込みﾃﾞｨｾｰﾌﾞﾙ

    ADAveFlag = 0;     // 平均化終了ﾌﾗｸﾞ ｸﾘｱ

    SegBuf[3] = SegTable[ADch+1]|0x80;     // A/Dch ｾｯﾄ
    SegBuf[2] = SegTable[(data>>8)&0x03]; // A/D 値(HEX) 3 桁目 ｾｯﾄ
    SegBuf[1] = SegTable[(data>>4)&0x0f]; // A/D 値(HEX) 2 桁目 ｾｯﾄ
    SegBuf[0] = SegTable[data&0x0f];       // A/D 値(HEX) 1 桁目 ｾｯﾄ

    set_imask_ccr(0);  // 割り込みｲﾈｰﾌﾞﾙ
}


/************************************************************************
    キー入力チェック
************************************************************************/
void chk_key(void)
{
    KeyFlag = 0;
    switch(KeyNo){
        case 9:
            ADch = 0; break;
        case 10:
            ADch = 1; break;
        case 11:
            ADch = 2; break;
        case 6:
            ADch = 3; break;
    }
}


/************************************************************************
    タイマＺ イニシャライズ
************************************************************************/
void init_tmrz(void)
{
    P_TMRZ0.TCR0.BYTE   =    0x23;     // CLK=X'tal/8,GRA ｺﾝﾍﾟｱﾏｯﾁ TCNT ｸﾘｱ
    P_TMRZ0.TIORA0.BYTE=    0x88;     // GRA ｱｳﾄﾌﾟｯﾄｺﾝﾍﾟｱﾚｼﾞｽﾀ,端子出力禁止
    P_TMRZ0.GRA0        =    2500;     // 周期設定 / 1msec
    P_TMRZ0.TIER0.BIT.IMIEA = 1; // TCNT=GRA ｲﾝﾀﾗﾌﾟﾄ ｲﾈｰﾌﾞﾙ
}


/************************************************************************
    タイマＢ１ イニシャライズ
************************************************************************/
void init_tmrb1(void)
{
    P_TMRB1.TMB1.BYTE   =    0xfb;          // オートリロード,X'tal/256
    P_TMRB1.TCB1        =    256-195;  // 2.496msec
}


/************************************************************************
    Ｉ／Ｏポート イニシャライズ
```

```
*************************************************************************/
void init_port(void)
{
    P_PORT.PMR1.BYTE   =   0x00;    //P10,11,14-17,22,72 使用
    P_PORT.PCR1.BYTE   =   0xf0;    //P10-12 In / P14-17 Out
    P_PORT.PUCR1.BYTE  =   0x00;    //プルアップ MOS 使用しない
    P_PORT.PDR1.BYTE   =   0x00;    //P14-17 = Low

    P_PORT.PCR2.BYTE   =   0x18;    //P20-22 In / P23,24 Out
    P_PORT.PDR2.BYTE   =   0x00;    //P23,24 = Low
    P_PORT.PMR3.BYTE   =   0x00;    //P23,24 CMOS Out

    P_PORT.PCR6.BYTE   =   0xFF;    //P60-67 Out
    P_PORT.PDR7.BYTE   =   0x00;    //P60-67 = Low

    P_PORT.PCR7.BYTE   =   0x77;    //P70-72,74-76 Out
    P_PORT.PDR7.BYTE   =   0x00;    //P70-72,74-76 = Low
}


/*************************************************************************
    割込みコントローラ イニシャライズ
*************************************************************************/
void init_int(void)
{
    P_INT.IRR2.BIT.IRRTB1   =   0;   //タイマ B1 割込み要求フラグクリア
    P_INT.IENR2.BIT.IENTB1  =   1;   //タイマ B1 割込み要求イネーブル
}


/*************************************************************************
    Ａ／Ｄ変換器 イニシャライズ
*************************************************************************/
void init_adc(void)
{
    P_AD.ADCSR.BYTE    =   0x53;    // ｽｷｬﾝﾓｰﾄﾞ AN0-3,134 ｽﾃｰﾄ,ｲﾝﾀﾗﾌﾟﾄｲﾈｰﾌﾞﾙ
}


/*************************************************************************
    End of Dynamic Scan Program.
*************************************************************************/
```

第3章 DA コンバータ

1. DA コンバータ AD5300
2. 1 変換プログラム
3. 任意周波数の矩形波出力

前章でアナログ値をデジタル値に変換する A/D コンバータについて学習しました。この章で取り上げる D/A コンバータは A/D コンバータの逆の働きをします。つまり、2 進数（デジタル値）で出力値を入力するとその値に相当した電圧（アナログ値）を出力するのです。

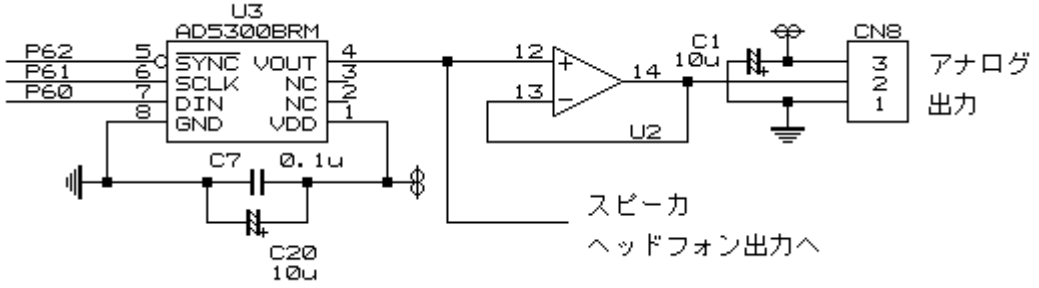
I/F トレーニングユニットで使用している D/A コンバータ AD5300 はシリアルインターフェース、レール to レール出力、つまり 0～5V 出力が可能な高性能 8bitD/A コンバータです。D/A コンバータの仕組みについては他の文献に譲るとして、ここでは AD5300 を使用したプログラムを示していきます。

1. D/A コンバータ AD5300

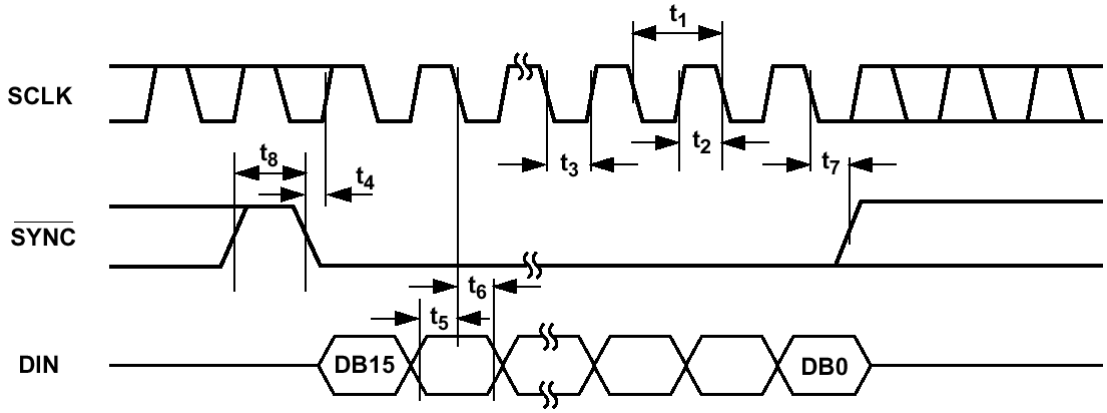
D/A 変換のプログラムを作成する前に、使用する D/A コンバータ AD5300 がどのようなものなのか知らなければなりません。まず AD5300 がどのようなものなのかをここで示しておきます。

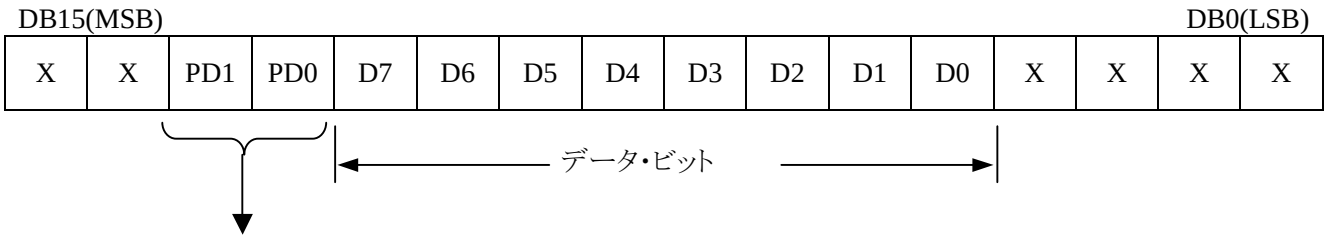
冒頭でも紹介したように AD5300 はシリアルインターフェース、レール to レール出力の 8bitD/A コンバータです。出力電圧範囲はレール to レールですので 0～電源電圧（VDD）つまり 0～5V で、分解能が 8bit ですから約 0.02V ステップで出力をコントロールすることが出来ます。

次にシリアルインターフェースについて調べてみましょう。AD5300 部の回路図を示します。



回路図から見て分かる様に AD5300 のコントロール端子と TK-3687 のポート 6 が接続されていますのでポート 6 をコントロールして AD5300 にデータを書き込むわけです。次に AD5300 のシリアル書き込み動作と入力するデータフォーマットを示します。





PD1	PD0	動作モード	
0	0	通常動作	
0	1	1kΩ経由で GND へ	パワーダウン・モード
1	0	100kΩ経由で GND へ	
1	1	3 状態	

通常は$\overline{\text{SYNC}}$(P62)・SCLK(P61)を High にしておきます。書き込み時には、$\overline{\text{SYNC}}$を Low にし DIN(P60)にデータをセットして、SCLK を立ち下げて 1bit 書き込みこれを 16 回繰り返します。16 クロック終えたら$\overline{\text{SYNC}}$を High に戻します。出力は$\overline{\text{SYNC}}$が Low で 16 番目の SCLK の立ち下がりで更新されます。16 番目の SCLK 立ち下がり前に$\overline{\text{SYNC}}$を High にしてしまうとそれまで書き込んでいたデータは無効になってしまいます。

動作モードは通常動作(PD1,0＝00)です。今回作成するプログラムではパワーダウン・モードは使用しませんので PD1,0 には 00 をセットします。

それでは次章から実際にプログラムをしてみましょう。

2. 1 変換プログラム

基板上のスイッチから出力レベルを 0～255 で指定し、D/A コンバータで指定されたレベルの電圧を出力するプログラムを作成します。10 進数で入力し Enter キーに相当するスイッチで確定、この時入力された値が 256 以上の場合は 255 として受け付けます。

まずは D/A コンバータに 1 データ出力する関数を作成しましょう。概要は前章の書き込み動作で述べた通りで、通常は$\overline{\text{SYNC}}$・SCLK を High にておき、書き込み時になったら$\overline{\text{SYNC}}$を Low にセットします。次に DIN に送信するデータをセットして SCLK を立ち下げ 1bit か書き込みます。書き込んだら SCLK を High に戻し、次の書き込むデータを DIN にセットし再度 SCLK を立ち下げます。これを 16 回繰り返し、16bit 書き込みが終了したら$\overline{\text{SYNC}}$・SCLK を High に戻し終了です。これらの操作は全てポートで行いますので割り付けられている P60～P62 の各ビットをセット又はリセットします。

以下に 1D/A 変換のプログラムを示します。

```
/************************************************************
    D／Aコンバータ出力 ／ 出力時間＝31.3μsec
************************************************************/
// for 文使用
void dac_out(char data)
{
    int      i;
    char p6data;

    // DAC ｽﾀｰﾄｱｯﾌﾟ : DIN=Low , /SYNC=Low , SCLK=High
    P_PORT.PDR6.BYTE = (P_PORT.PDR6.BYTE & 0xf8) | 0x02;

    // DB15-12 書込み : DB15-14=xx , DB13-12=PD1-0=00=通常動作
    for(i=0;i<4;i++){
        P_PORT.PDR6.BIT.P61 = 0;     // SCLK=Low
        P_PORT.PDR6.BIT.P61 = 1;     // SCLK=High
        }

    // ﾃﾞｰﾀ書込み : DB11-4=ﾃﾞｰﾀ
    for(i=0;i<8;i++){
        if(data & 0x80)         // DIN 出力
            P_PORT.PDR6.BIT.P60 = 1;
        else
            P_PORT.PDR6.BIT.P60 = 0;
        data <<= 1;
        P_PORT.PDR6.BIT.P61 = 0;     // SCLK=Low
        P_PORT.PDR6.BIT.P61 = 1;     // SCLK=High
        }

    // DB3-0 書込み : DB3-0=xx
    P_PORT.PDR6.BIT.P60 = 0;         // DIN=Low
    for(i=0;i<4;i++){
        P_PORT.PDR6.BIT.P61 = 0;     // SCLK=Low
        P_PORT.PDR6.BIT.P61 = 1;     // SCLK=High
        }

    // 終了
    P_PORT.PDR6.BIT.P62 = 1;         // /SYNC=High
}
```

それでは 1D/A 変換の関数とキー入力を組み合わせてプログラムを作成しましょう。

まずキーの割り付けを決めます。10 進数で出力レベルを指定するので 0～9 までの数字キーが必要です。また入力した値を確定する SET キーも必要です。I/F トレーニングユニットにはキーが全部で 12 個載っていますのでこれで 11 個割り付けが決まりました。まだ割り付けられていないキーが 1 つありますので、これは入力をクリア（キャンセル）するキーします。これで全て割り付けられました。キーの並びは以下のようにすることとします。

SW9	SW10	SW11
1	2	3
SW6	SW7	SW8
4	5	6
SW3	SW4	SW5
7	8	9
SW0	SW1	SW2
CLR	0	SET

次に表示です。出力の範囲は 000～255 までですので 3 桁あれば事足りますので、下 3 桁を出力レベルの表示として使用します。4 桁目を遊ばせておくのも無駄なので d.p.を利用して現在の状態を表しましょう。D/A 出力中には d.p.を点灯、数字キーでの出力指定中は d.p.消灯とします。通常は 4 桁目の d.p.を点灯し現在の出力レベルを下 3 桁に表示します。数字キーが押されたら全桁表示をクリアし押された番号を下桁に表示します。以後数字キーが押される度に表示を左シフトして押された番号を下桁に表示していきます。SET キーが押されたら 4 桁目 d.p.を点灯し出力状態を表します。但し、入力された値が 256 以上の場合は強制的に 255 とします。CLR キーが押されたら現在の出力レベルを表示します。

以上を踏まえて作成したプログラムのリストを以下に示します。

```
/************************************************************/
/*
/*  FILE        :dac_10key.c
/*  DATE        :Mon, Apr 05, 2004
/*  DESCRIPTION :Main Program
/*  CPU TYPE    :H8/3687
/*
/*  This file is generated by Hitachi Project Generator (Ver.2.1).
/*               Programed by Toyo-linx,Co.,Ltd. / Y.Furukawa.
/*
/************************************************************/
/*
■概要
  基板上のキーから入力した値を D/A 変換し出力します。10 進数で入力し、入力範囲は 0～255 です。256 以上の入力は 255 として扱われます。入力した値を出力するには SET キーを、キャンセルは CLR キーを押します。キーの割り付けは以下の通りです。
    SW9 [1]   SW10[2]   SW11[3]
    SW6 [4]   SW7 [5]   SW8 [6]
    SW3 [7]   SW4 [8]   SW5 [9]
    SW0 [CLR]  SW1 [0]   SW2 [SET]
*/

/************************************************************
     履歴
************************************************************/
/*
2004-04-05     作成開始
```

```
*/


/************************************************************
    インクルードファイル
************************************************************/
#include <machine.h>
#include "iodefine.h"


/************************************************************
    定数の定義（直接指定）
************************************************************/
// 7セグメントLEDに関係する定数 ----------------------------------------
#define       KETA          4         //7セグメントLEDの桁数

// その他 ---------------------------------------------------------------
#define       OK                0         //戻り値
#define       NG                -1        //戻り値


/************************************************************
    定数の定義（ROM）
************************************************************/
// 7セグメントデータ変換テーブル ----------------------------------------
const unsigned charSegTable[16] = {0x3f,00x6,0x5b,0x4f,   //0,1,2,3
                                   0x66,0x6d,0x7d,0x07,   //4,5,6,7
                                      0x7f,0x67,0x77,0x7c,    //8,9,A,B
                                      0x39,0x5e,0x79,0x71};   //C,D,E,F


/************************************************************
    スタティック変数の定義とイニシャライズ
    ここでイニシャライズしていない変数はプログラム中で行なっている
************************************************************/
// 7セグメントLEDスキャンに関する変数 -----------------------------
unsigned char  SegBuf[4] = {0x3f   //7セグメントLEDバッファ
                           ,0x3f   //（初期表示）
                              ,0x3f
                              ,0x80};
unsigned char  ScanPnt = 0;       //どのスキャンラインか
                                        //（キースキャンと兼用）

// キースキャンに関する変数 ---------------------------------------------
unsigned char  Key1stBuf[4];      //キー1stリード
unsigned char  Key2ndBuf[4];      //キー2ndリード
unsigned char  KeyBuf[4];             //1st=2ndになったキーデータ
unsigned char  KeyReadFlag = 0;   //キーリードフラグ
                                        // 0:1stリード
                                        // 1:2ndリード
int               KeyNo = 0;              //キー番号
int               KeyNoOld = -1;      //前回のキー番号
unsigned char  KeyFlag = 0;       //キー入力フラグ
                                        // 1:キー入力あり

// D/Aコンバータに関する変数 --------------------------------------------
unsigned char  DACdata = 0x00;         // D/Aｺﾝﾊﾞｰﾀ ﾃﾞｰﾀ
```

```
int               IDflag = 0;             // 入力ﾌﾗｸﾞ
int               ID1stflag = 0;     // 入力中ﾌﾗｸﾞ
unsigned char  IDbuf[3]           // 現在の D/A 値
                       ={0,0,0};
unsigned char  IDbufBAK[3]             // D/A 値のﾊﾞｯｸｱｯﾌﾟ
                       ={0,0,0};


/*********************************************************
    関数の定義
*********************************************************/
void          init_int(void);          // 割り込みｺﾝﾄﾛｰﾗ ｲﾆｼｬﾗｲｽﾞ
void          init_port(void);    // I/O ﾎﾟｰﾄ ｲﾆｼｬﾗｲｽﾞ
void          init_tmrb1(void);   // ﾀｲﾏ B1 ｲﾆｼｬﾗｲｽﾞ
void          main(void);              // ﾒｲﾝﾌﾟﾛｸﾞﾗﾑ
void          scan(void);              // ｷｰ&LED ｽｷｬﾝ (ﾀｲﾏ B1 割り込み)
void          disp(void);              // LED ｽｷｬﾝ
void          keyin(void);        // ｷｰｽｷｬﾝ
void          set_disp(void);          // 表示ﾃﾞｰﾀ ｾｯﾄ
void          chk_key(void);      // ｷｰ入力ﾁｪｯｸ
void          dac_out(char);      // D/A ｺﾝﾊﾞｰﾀ出力
void          dac_out_hs(char);   // D/A ｺﾝﾊﾞｰﾀ出力(HighSpeed)


/*********************************************************
    メインプログラム
*********************************************************/
void main(void)
{
//----- イニシャライズ -------------------------------------------------
    init_tmrb1();       // ﾀｲﾏ B1 ｲﾆｼｬﾗｲｽﾞ
    init_port();        // I/O ﾎﾟｰﾄ ｲﾆｼｬﾗｲｽﾞ
    init_int();              // 割り込み ｲﾆｼｬﾗｲｽﾞ
    dac_out(DACdata);   // D/A ｲﾆｼｬﾙ出力
    set_imask_ccr(0);   // 割り込みｲﾈｰﾌﾞﾙ

//----- メインループ ---------------------------------------------------
    while(1){
        if (KeyFlag == 1){           // ｷｰ入力 ﾁｪｯｸ
            KeyFlag = 0;             // ﾌﾗｸﾞ ｸﾘｱ
            chk_key();                    // ｷｰ動作
            set_disp();                   // 表示 ｾｯﾄ
            if (IDflag == 1){
                IDflag = 0;              // ﾌﾗｸﾞ ｸﾘｱ
                dac_out(DACdata);   // D/A 出力
            }
        }
    }
//---------------------------------------------------------------------
}


/*********************************************************
    キー&LED スキャン (タイマ B1 割り込み)
*********************************************************/
#pragma regsave (scan)
void scan(void)
{
```

```
    disp();                          // 7segLED ｽｷｬﾝ
    keyin();                      // ｷｰ ｽｷｬﾝ
    ScanPnt++; if (ScanPnt>(KETA-1)) ScanPnt=0;//ｽｷｬﾝｶｳﾝﾀの更新
}


/*************************************************************
    Ｄ／Ａコンバータ出力 ／ 出力時間＝31.3μsec
*************************************************************/
// for 文使用
void dac_out(char data)
{
    int       i;
    char p6data;

    // DAC ｽﾀｰﾄｱｯﾌﾟ : DIN=Low , /SYNC=Low , SCLK=High
    P_PORT.PDR6.BYTE = (P_PORT.PDR6.BYTE & 0xf8) | 0x02;

    // DB15-12 書込み : DB15-14=xx , DB13-12=PD1-0=00=通常動作
    for(i=0;i<4;i++){
        P_PORT.PDR6.BIT.P61 = 0;     // SCLK=Low
        P_PORT.PDR6.BIT.P61 = 1;     // SCLK=High
        }

    // ﾃﾞｰﾀ書込み : DB11-4=ﾃﾞｰﾀ
    for(i=0;i<8;i++){
        if(data & 0x80)          // DIN 出力
            P_PORT.PDR6.BIT.P60 = 1;
        else
            P_PORT.PDR6.BIT.P60 = 0;
        data <<= 1;
        P_PORT.PDR6.BIT.P61 = 0;     // SCLK=Low
        P_PORT.PDR6.BIT.P61 = 1;     // SCLK=High
        }

    // DB3-0 書込み : DB3-0=xx
    P_PORT.PDR6.BIT.P60 = 0;          // DIN=Low
    for(i=0;i<4;i++){
        P_PORT.PDR6.BIT.P61 = 0;     // SCLK=Low
        P_PORT.PDR6.BIT.P61 = 1;     // SCLK=High
        }

    // 終了
    P_PORT.PDR6.BIT.P62 = 1;          // /SYNC=High
}


/*************************************************************
    LED スキャン
*************************************************************/
void disp(void)
{
      ｛
      ｝
}
```

```
/************************************************************
    キー スキャン
************************************************************/
void keyin(void)
{
      ｛
      ｝
}


/************************************************************
    キー入力チェック
************************************************************/
void chk_key(void)
{
    int               i;
    int               j;
    int               data;
    unsigned char key;

    switch (KeyNo){
    // CLR キー
        case 0:
            if (ID1stflag == 1){
                ID1stflag = 0;              // ﾌﾗｸﾞ ｸﾘｱ
                for (i=0;i<3;i++){
                    IDbuf[i] = IDbufBAK[i]; // 今までの値を戻す
                    IDbufBAK[i] = 0;        // ﾊﾞｯｸｱｯﾌﾟﾊﾞｯﾌｧ ｸﾘｱ
                }
            }
            return;                             // 終了
    // SET キー
        case 2:
            ID1stflag = 0;                  // ﾌﾗｸﾞ ｸﾘｱ
            data = 0;                       // D/A 出力値 ｸﾘｱ
            for (i=0,j=1;i<3;i++,j=j*10){
                data = data+(IDbuf[i]*j);   // D/A 出力値 ｾｯﾄ
            }
            if (data < 256)                     // 入力値判定
                DACdata = data;                 // 255 以内
            else{
                DACdata = 255;              // 256 以上
                IDbuf[0] = 5;               //  256 以上は 255 とする
                IDbuf[1] = 5;
                IDbuf[2] = 2;
            }
            IDflag = 1;                         // ﾌﾗｸﾞ ｾｯﾄ
            return;                             // 終了
    }

    // １０キー
    if (ID1stflag==0){              // 初回入力？
        ID1stflag = 1;              // ﾌﾗｸﾞ ｾｯﾄ
        for (i=0;i<3;i++){
            IDbufBAK[i] = IDbuf[i]; // 現在値ﾊﾞｯｸ ｱｯﾌﾟ
            IDbuf[i] = 0;           // 入力ﾊﾞｯﾌｧ ｸﾘｱ
        }
```

```
    }
    switch (KeyNo){                         // 押されたｷｰ番号を求める
        case 1:                             // ｷｰ 0
            key = 0;
            break;
        case 9:                             // ｷｰ 1
        case 10:                        // ｷｰ 2
        case 11:                        // ｷｰ 3
            key = KeyNo-8;
            break;
        case 6:                             // ｷｰ 4
        case 7:                             // ｷｰ 5
        case 8:                             // ｷｰ 6
            key = KeyNo-2;
            break;
        case 3:                             // ｷｰ 7
        case 4:                             // ｷｰ 8
        case 5:                             // ｷｰ 9
            key = KeyNo+4;
            break;
    }

    IDbuf[2] = IDbuf[1];                // ﾃﾞｰﾀｼﾌﾄ
    IDbuf[1] = IDbuf[0];
    IDbuf[0] = key;                         // 今押された番号をｾｯﾄ
}


/*************************************************************
    表示データセット
*************************************************************/
void set_disp(void)
{
    int  i;

    if (ID1stflag == 1)             // 4桁目 ｾｯﾄ
        SegBuf[3] = 0;              // 入力中
    else
        SegBuf[3] = 0x80;           // 出力中

    for (i=0;i<3;i++){              // D/A値 ｾｯﾄ
        SegBuf[i] = SegTable[IDbuf[i]];
    }
}


/*************************************************************
    タイマＢ１ イニシャライズ
*************************************************************/
void init_tmrb1(void)
{
    P_TMRB1.TMB1.BYTE  =    0xfb;           // ｵｰﾄﾘﾛｰﾄﾞ,X'tal/256
    P_TMRB1.TCB1       =    256-195;  // 2.496msec
}


/*************************************************************
    Ｉ／Ｏポート イニシャライズ
```

```
************************************************************/
void init_port(void)
{
    P_PORT.PMR1.BYTE   =   0x00;    // P10,11,14-17,22,72 使用
    P_PORT.PCR1.BYTE   =   0xf0;    // P10-12 In / P14-17 Out
    P_PORT.PUCR1.BYTE  =   0x00;    // ﾌﾟﾙｱｯﾌﾟ MOS 使用しない
    P_PORT.PDR1.BYTE   =   0x00;    // P14-17 = Low

    P_PORT.PCR2.BYTE   =   0x18;    // P20-22 In / P23,24 Out
    P_PORT.PDR2.BYTE   =   0x00;    // P23,24 = Low
    P_PORT.PMR3.BYTE   =   0x00;    // P23,24 CMOS Out

    // P60:DIN  P61:SCLK  P62:/SYNC  P63-67:output of check
    P_PORT.PCR6.BYTE   =   0xFF;    // P60-67 Out
    P_PORT.PDR6.BYTE   =   0x06;    // P61,P62=High / P60,P63-P67=Low

    P_PORT.PCR7.BYTE   =   0x77;    // P70-72,74-76 Out
    P_PORT.PDR7.BYTE   =   0x00;    // P70-72,74-76 = Low
}


/************************************************************
    割込みコントローラ イニシャライズ
************************************************************/
void init_int(void)
{
    P_INT.IRR2.BIT.IRRTB1   =   0;  // ﾀｲﾏ B1 割込み要求ﾌﾗｸﾞ ｸﾘｱ
    P_INT.IENR2.BIT.IENTB1  =   1;  // ﾀｲﾏ B1 割込み要求ｲﾈｰﾌﾞﾙ
}


/************************************************************
    End of Dynamic Scan Program.
************************************************************/
```

それではD/A変換が行われているか、テスタで確認してみましょう。キーで0～255までの出力レベルをセットし、CN8の2番ピンの電圧を測ってみましょう。出力レベルが000の時は0Vが、出力レベルが255の時は5VがCN8に出ていると思います。又出力レベルを128に指定すると2.5V出ていればD/A変換は正常に行われています。

3. 任意周波数の矩形波出力

次に応用として任意の周波数の矩形波を D/A コンバータから出力します。矩形波ですので D/A コンバータにセットするデータは 0x00 か 0xFF のみです。タイマ Z を使用し、出力したい周波数の 1/2 周期で割り込みをかけ出力を反転します。周波数は 1kHz を基準に上は 2kHz・4kHz・8kHz・16kHz、下は 500Hz・250Hz・125Hz・62.5Hz としました。キーにこれらの周波数を割り付け、対応するキーが押されたらその周波数を出力し、7 セグメント LED に出力している周波数を表示します。尚、4 桁した表示が無いので 62.5Hz～500Hz までは Hz 表示とし 2 桁目に小数点を打って“62.5”～“500.0”と表し、1kHz～16kHz は kHz 表示とし 1 桁目の d.p.で kHz を表現することにします。表示は“1.”～“16.”となります。キーの割り付けを以下に示します。

SW9	SW10	SW11
62.5Hz	125Hz	250Hz
SW6	SW7	SW8
500Hz	1kHz	2kHz
SW3	SW4	SW5
4kHz	8kHz	16kHz
SW0	SW1	SW2
-	-	-

キーが押されたらそのキーに対応する周波数の 1/2 周期でカウントアップするようにタイマ Z をセットしリスタートし出力する周波数を表示にセットします。周期を作る為にタイマ Z の TCNT にカウント値をそれぞれセットしますがクロックを変更しないと TCNT だけでは全ての周波数はカバーできません。そこで 62.5Hz～500Hz までは CLK/8 を、1kHz～16kHz は CLK をクロックとして選択します。

尚、前回紹介した 1D/A 変換のプログラムは for 文を用いていましたが、これでは処理時間が長く 16kHz の矩形波を出力することが出来ませんでした。そこでこのプログラムでは for 文を用いずに記述しています。プログラムは長くなってしまいますが for 文を用いるよりも 10μsec 程早く D/A コンバータにデータを書き込むことが出来ます。

矩形波出力のプログラムリストを次ページに示します。

```
/************************************************************/
/*
/*  FILE        :dac.c
/*  DATE        :Tue, Mar 02, 2004
/*  DESCRIPTION :Main Program
/*  CPU TYPE    :H8/3687
/*
/*  This file is generated by Hitachi Project Generator (Ver.2.1).
/*               programed by Toyo-linx,Co.,Ltd. / Y.Furukawa.
/*
/************************************************************/
/*
■概要
  矩形波をＤ／Ａコンバータで出力します。出力周波数は９段階で周波数の切り替えはスイッチで行います。出力周波数とキー割り付けは次の通りです。
出力周波数：
[ 62.5Hz,125Hz,250Hz,500Hz,1kHz,2kHz,4kHz,8kHz,16kHz ]
割り付け   :[ SW9(1)=62.5Hz / SW10(2)=125Hz / SW11(3)=250Hz ]
            [ SW6(4)= 500Hz / SW7 (5)= 1kHz / SW8 (6)= 2kHz ]
            [ SW3(7)=  4kHz / SW4 (8)= 8kHz / SW5 (9)=16kHz ]
出力している周波数は 7segLED に表示されます。尚、1000Hz 以上は kHz 表示とし、一桁目にドットを表示して kHz を表します（例：[  16.] = 16kHz ）。
*/

/************************************************************
    履歴
************************************************************/
/*
2004-03-02    作成開始
    03-03         D/A ｺﾝﾊﾞｰﾀより矩形波出力　出力周波数 9 段階（62.5Hz～16000Hz）
                  スイッチで周波数変更　7segLED へ出力周波数表示
     03-04    ﾀｲﾏ B1 多重割り込み許可ﾀｲﾐﾝｸﾞ変更
                  ｽｷｬﾝと D/A 出力ﾀｲﾐﾝｸﾞの一致による D/A 出力の揺らぎを解消
     04-02    D/A 出力 高速版を"dac_out_hs"として分離 その他コメント追加
*/


/************************************************************
    インクルードファイル
************************************************************/
#include  <machine.h>
#include  "iodefine.h"


/************************************************************
    定数の定義（直接指定）
************************************************************/
// 7 セグメント LED に関係する定数 ----------------------------------------
#define       KETA      4     //7 セグメント LED の桁数

// その他 ----------------------------------------------------------------
#define       OK            0     //戻り値
#define       NG            -1    //戻り値
```

```
/************************************************************
    定数の定義（ROM）
************************************************************/
// 7セグメントデータ変換テーブル ---------------------------------------
const unsigned charSegTable[16] = {0x3f,00x6,0x5b,0x4f,  //0,1,2,3
                                   0x66,0x6d,0x7d,0x07,  //4,5,6,7
                                      0x7f,0x67,0x77,0x7c,    //8,9,A,B
                                      0x39,0x5e,0x79,0x71};   //C,D,E,F


/************************************************************
    スタティック変数の定義とイニシャライズ
    ここでイニシャライズしていない変数はプログラム中で行なっている
************************************************************/
// 7セグメントLEDスキャンに関する変数 ------------------------------
unsigned char  SegBuf[4] = {0x86  //7ｾｸﾞﾒﾝﾄLEDﾊﾞｯﾌｧ
                           ,0x00  //（初期表示）
                           ,0x00
                           ,0x00};
unsigned char  ScanPnt = 0;       //どのスキャンラインか
                                  //（キースキャンと兼用）

// キースキャンに関する変数 ----------------------------------------
unsigned char  Key1stBuf[4];      //キー1stリード
unsigned char  Key2ndBuf[4];      //キー2ndリード
unsigned char  KeyBuf[4];             //1st=2ndになったｷｰﾃﾞｰﾀ
unsigned char  KeyReadFlag = 0;   //キーリードフラグ
                                  // 0:1stリード
                                  // 1:2ndリード
int            KeyNo = 0;             //キー番号
int            KeyNoOld = -1;         //前回のキー番号
unsigned char  KeyFlag = 0;       //キー入力フラグ
                                  // 1:キー入力あり

// D/Aコンバータに関する変数 ---------------------------------------
char           DACdata = 0x00;    // D/Aｺﾝﾊﾞｰﾀ ﾃﾞｰﾀ


/************************************************************
    関数の定義
************************************************************/
void           init_int(void);         // 割り込みｺﾝﾄﾛｰﾗｲﾆｼｬﾗｲｽﾞ
void           init_port(void);        // I/Oﾎﾟｰﾄ ｲﾆｼｬﾗｲｽﾞ
void           init_tmrz(void);        // ﾀｲﾏZ ｲﾆｼｬﾗｲｽﾞ
void           init_tmrb1(void);  // ﾀｲﾏB1 ｲﾆｼｬﾗｲｽﾞ
void           main(void);             // ﾒｲﾝﾌﾟﾛｸﾞﾗﾑ
void           scan(void);             // ｷｰ&LEDｽｷｬﾝ（ﾀｲﾏB1割り込み）
void           disp(void);             // LEDｽｷｬﾝ
void           keyin(void);       // ｷｰｽｷｬﾝ
void           chk_key(void);     // ｷｰ入力ﾁｪｯｸ
void           int_tz1(void);     // ﾀｲﾏZ1 割り込み処理
void           dac_out(char);     // D/Aｺﾝﾊﾞｰﾀ出力
void           dac_out_hs(char);  // D/Aｺﾝﾊﾞｰﾀ出力(HighSpeed)


/************************************************************
    メインプログラム
************************************************************/
```

```
void main(void)
{
//----- イニシャライズ -------------------------------------------------
     init_tmrz();   // ﾀｲﾏ Z ｲﾆｼｬﾗｲｽﾞ
     init_tmrb1(); // ﾀｲﾏ B1 ｲﾆｼｬﾗｲｽﾞ
     init_port();   // I/O ﾎﾟｰﾄ ｲﾆｼｬﾗｲｽﾞ
     init_int();         // 割り込み ｲﾆｼｬﾗｲｽﾞ

     P_TMRZ.TSTR.BIT.STR1 = 1;    // ﾀｲﾏ Z TCNT_1 ｶｳﾝﾄ開始

     set_imask_ccr(0);  //割り込みｲﾈｰﾌﾞﾙ

//----- メインループ ---------------------------------------------------
     while(1){
          if (KeyFlag == 1)  chk_key();     // ｷｰ入力ﾁｪｯｸ
     }
//---------------------------------------------------------------------
}


/*************************************************************
     タイマＺ１割り込み処理
*************************************************************/
void int_tz1(void)
{
     DACdata ^= 0xff;         // 出力ﾃﾞｰﾀ反転
     dac_out_hs(DACdata);     // D/A 出力(HighSpeed)
//   dac_out(DACdata);        // D/A 出力
}


/*************************************************************
     Ｄ／Ａコンバータ出力 ／ 出力時間＝31.3μsec
*************************************************************/
// for 文使用
void dac_out(char data)
{
     int       i;
     char p6data;

     // DAC ｽﾀｰﾄｱｯﾌﾟ : DIN=Low , /SYNC=Low , SCLK=High
     P_PORT.PDR6.BYTE = (P_PORT.PDR6.BYTE & 0xf8) | 0x02;

     // DB15-12 書込み : DB15-14=xx , DB13-12=PD1-0=00=通常動作
     for(i=0;i<4;i++){
          P_PORT.PDR6.BIT.P61 = 0;     // SCLK=Low
          P_PORT.PDR6.BIT.P61 = 1;     // SCLK=High
          }

     // ﾃﾞｰﾀ書込み : DB11-4=ﾃﾞｰﾀ
     for(i=0;i<8;i++){
          if(data & 0x80)          // DIN 出力
               P_PORT.PDR6.BIT.P60 = 1;
          else
               P_PORT.PDR6.BIT.P60 = 0;
          data <<= 1;
          P_PORT.PDR6.BIT.P61 = 0;     // SCLK=Low
          P_PORT.PDR6.BIT.P61 = 1;     // SCLK=High
```

```
        }

    // DB3-0 書込み : DB3-0=xx
    P_PORT.PDR6.BIT.P60 = 0;          // DIN=Low
    for(i=0;i<4;i++){
        P_PORT.PDR6.BIT.P61 = 0;      // SCLK=Low
        P_PORT.PDR6.BIT.P61 = 1;      // SCLK=High
        }

    // 終了
    P_PORT.PDR6.BIT.P62 = 1;          // /SYNC=High
}


/************************************************************
    Ｄ／Ａコンバータ出力（HighSpeed）／ 出力時間＝19.8μsec
************************************************************/
// 処理速度優先 for 文は使用せず
void dac_out_hs(char data)
{
    int       i;
    char p6data;

    p6data = P_PORT.PDR6.BYTE;

    // DAC ｽﾀｰﾄｱｯﾌﾟ : DIN=Low , /SYNC=Low , SCLK=High
    p6data = (p6data & 0xf8) | 0x02;
    P_PORT.PDR6.BYTE = p6data;

    // DB15-12 書込み : DB15-14=xx , DB13-12=PD1-0=00=通常動作
    p6data &= 0xfd;
    P_PORT.PDR6.BYTE = p6data;   // SCLK=Low        DB15
    p6data |= 0x02;
    P_PORT.PDR6.BYTE = p6data;   // SCLK=High
    p6data &= 0xfd;
    P_PORT.PDR6.BYTE = p6data;   // SCLK=Low        DB14
    p6data |= 0x02;
    P_PORT.PDR6.BYTE = p6data;   // SCLK=High
    p6data &= 0xfd;
    P_PORT.PDR6.BYTE = p6data;   // SCLK=Low        DB13
    p6data |= 0x02;
    P_PORT.PDR6.BYTE = p6data;   // SCLK=High
    p6data &= 0xfd;
    P_PORT.PDR6.BYTE = p6data;   // SCLK=Low        DB12
    p6data |= 0x02;
    P_PORT.PDR6.BYTE = p6data;   // SCLK=High

    // ﾃﾞｰﾀ書込み : DB11-4=ﾃﾞｰﾀ
    p6data = (p6data & 0xfe) | (0x01 & (data=rotlc(1,data)));
    P_PORT.PDR6.BYTE = p6data;   // DIN 出力        DB11
    p6data &= 0xfd;
    P_PORT.PDR6.BYTE = p6data;   // SCLK=Low
    p6data |= 0x02;
    P_PORT.PDR6.BYTE = p6data;   // SCLK=High
    p6data = (p6data & 0xfe) | (0x01 & (data=rotlc(1,data)));
    P_PORT.PDR6.BYTE = p6data;   // DIN 出力        DB10
    p6data &= 0xfd;
    P_PORT.PDR6.BYTE = p6data;   // SCLK=Low
```

```
    p6data |= 0x02;
    P_PORT.PDR6.BYTE = p6data;   // SCLK=High
    p6data = (p6data & 0xfe) | (0x01 & (data=rotlc(1,data)));
    P_PORT.PDR6.BYTE = p6data;   // DIN 出力         DB9
    p6data &= 0xfd;
    P_PORT.PDR6.BYTE = p6data;   // SCLK=Low
    p6data |= 0x02;
    P_PORT.PDR6.BYTE = p6data;   // SCLK=High
    p6data = (p6data & 0xfe) | (0x01 & (data=rotlc(1,data)));
    P_PORT.PDR6.BYTE = p6data;   // DIN 出力         DB8
    p6data &= 0xfd;
    P_PORT.PDR6.BYTE = p6data;   // SCLK=Low
    p6data |= 0x02;
    P_PORT.PDR6.BYTE = p6data;   // SCLK=High
    p6data = (p6data & 0xfe) | (0x01 & (data=rotlc(1,data)));
    P_PORT.PDR6.BYTE = p6data;   // DIN 出力         DB7
    p6data &= 0xfd;
    P_PORT.PDR6.BYTE = p6data;   // SCLK=Low
    p6data |= 0x02;
    P_PORT.PDR6.BYTE = p6data;   // SCLK=High
    p6data = (p6data & 0xfe) | (0x01 & (data=rotlc(1,data)));
    P_PORT.PDR6.BYTE = p6data;   // DIN 出力         DB6
    p6data &= 0xfd;
    P_PORT.PDR6.BYTE = p6data;   // SCLK=Low
    p6data |= 0x02;
    P_PORT.PDR6.BYTE = p6data;   // SCLK=High
    p6data = (p6data & 0xfe) | (0x01 & (data=rotlc(1,data)));
    P_PORT.PDR6.BYTE = p6data;   // DIN 出力         DB5
    p6data &= 0xfd;
    P_PORT.PDR6.BYTE = p6data;   // SCLK=Low
    p6data |= 0x02;
    P_PORT.PDR6.BYTE = p6data;   // SCLK=High
    p6data = (p6data & 0xfe) | (0x01 & (data=rotlc(1,data)));
    P_PORT.PDR6.BYTE = p6data;   // DIN 出力         DB4
    p6data &= 0xfd;
    P_PORT.PDR6.BYTE = p6data;   // SCLK=Low
    p6data |= 0x02;
    P_PORT.PDR6.BYTE = p6data;   // SCLK=High

    // DB3-0 書込み : DB3-0=xx
    p6data &= 0xfd;
    P_PORT.PDR6.BYTE = p6data;   // SCLK=Low         DB3
    p6data |= 0x02;
    P_PORT.PDR6.BYTE = p6data;   // SCLK=High
    p6data &= 0xfd;
    P_PORT.PDR6.BYTE = p6data;   // SCLK=Low         DB2
    p6data |= 0x02;
    P_PORT.PDR6.BYTE = p6data;   // SCLK=High
    p6data &= 0xfd;
    P_PORT.PDR6.BYTE = p6data;   // SCLK=Low         DB1
    p6data |= 0x02;
    P_PORT.PDR6.BYTE = p6data;   // SCLK=High
    p6data &= 0xfd;
    P_PORT.PDR6.BYTE = p6data;   // SCLK=Low         DB0
    p6data |= 0x02;
    P_PORT.PDR6.BYTE = p6data;   // SCLK=High

    // 終了
```

```
    p6data = (p6data & 0xf8) | 0x06;
    P_PORT.PDR6.BYTE = p6data;        // /SYNC=High
}


/************************************************************
    キー&LED スキャン（タイマ B1 割り込み）
************************************************************/
#pragma regsave (scan)
void scan(void)
{
      ｛
      ｝
}


/************************************************************
    LED スキャン
************************************************************/
void disp(void)
{
      ｛
      ｝
}


/************************************************************
    キー スキャン
************************************************************/
void keyin(void)
{
      ｛
      ｝
}


/************************************************************
    キー入力チェック
************************************************************/
void chk_key(void)
{
    KeyFlag = 0;
    switch (KeyNo){
        // 出力周波数変更 f=62.5Hz
        case 9:
            set_imask_ccr(1);  // 割り込みﾃﾞｨｾｰﾌﾞﾙ
            P_TMRZ.TSTR.BIT.STR1 = 0;    // ﾀｲﾏ Z TCNT_1 ｶｳﾝﾄ停止
            P_TMRZ1.TCR1.BYTE = 0x23;    // CLK=X'tal/8
            P_TMRZ1.GRA1 = 20000;        // GRA 設定
            P_TMRZ.TSTR.BIT.STR1 = 1;    // ﾀｲﾏ Z TCNT_1 ｶｳﾝﾄ開始
            SegBuf[3] = 0;               // 4 桁目 ｾｯﾄ
            SegBuf[2] = SegTable[6];          // 3 桁目 ｾｯﾄ 6
            SegBuf[1] = SegTable[2]|0x80;     // 2 桁目 ｾｯﾄ 2.
            SegBuf[0] = SegTable[5];          // 1 桁目 ｾｯﾄ 5
            set_imask_ccr(0);  // 割り込みｲﾈｰﾌﾞﾙ
```

```
        break;

    // 出力周波数変更 f=125Hz
    case 10:
        set_imask_ccr(1);  // 割り込みﾃﾞｨｾｰﾌﾞﾙ
        P_TMRZ.TSTR.BIT.STR1 = 0;    // ﾀｲﾏ Z TCNT_1 ｶｳﾝﾄ停止
        P_TMRZ1.TCR1.BYTE = 0x23;    // CLK=X'tal/8
        P_TMRZ1.GRA1 = 10000;        // GRA 設定
        P_TMRZ.TSTR.BIT.STR1 = 1;    // ﾀｲﾏ Z TCNT_1 ｶｳﾝﾄ開始
        SegBuf[3] = SegTable[1];          // 4 桁目 ｾｯﾄ 1
        SegBuf[2] = SegTable[2];          // 3 桁目 ｾｯﾄ 2
        SegBuf[1] = SegTable[5]|0x80;     // 2 桁目 ｾｯﾄ 5.
        SegBuf[0] = SegTable[0];          // 1 桁目 ｾｯﾄ 0
        set_imask_ccr(0);  // 割り込みｲﾈｰﾌﾞﾙ
        break;

    // 出力周波数変更 f=250Hz
    case 11:  // f=250Hz
        set_imask_ccr(1);  // 割り込みﾃﾞｨｾｰﾌﾞﾙ
        P_TMRZ.TSTR.BIT.STR1 = 0;    // ﾀｲﾏ Z TCNT_1 ｶｳﾝﾄ停止
        P_TMRZ1.TCR1.BYTE = 0x23;    // CLK=X'tal/8
        P_TMRZ1.GRA1 = 5000;         // GRA 設定
        P_TMRZ.TSTR.BIT.STR1 = 1;    // ﾀｲﾏ Z TCNT_1 ｶｳﾝﾄ開始
        SegBuf[3] = SegTable[2];          // 4 桁目 ｾｯﾄ 2
        SegBuf[2] = SegTable[5];          // 3 桁目 ｾｯﾄ 5
        SegBuf[1] = SegTable[0]|0x80;     // 2 桁目 ｾｯﾄ 0.
        SegBuf[0] = SegTable[0];          // 1 桁目 ｾｯﾄ 0
        set_imask_ccr(0);  // 割り込みｲﾈｰﾌﾞﾙ
        break;

    // 出力周波数変更 f=500Hz
    case 6:        // f=500Hz
        set_imask_ccr(1);  // 割り込みﾃﾞｨｾｰﾌﾞﾙ
        P_TMRZ.TSTR.BIT.STR1 = 0;    // ﾀｲﾏ Z TCNT_1 ｶｳﾝﾄ停止
        P_TMRZ1.TCR1.BYTE = 0x23;    // CLK=X'tal/8
        P_TMRZ1.GRA1 = 2500;         // GRA 設定
        P_TMRZ.TSTR.BIT.STR1 = 1;    // ﾀｲﾏ Z TCNT_1 ｶｳﾝﾄ開始
        SegBuf[3] = SegTable[5];          // 4 桁目 ｾｯﾄ 5
        SegBuf[2] = SegTable[0];          // 3 桁目 ｾｯﾄ 0
        SegBuf[1] = SegTable[0]|0x80;     // 2 桁目 ｾｯﾄ 0.
        SegBuf[0] = SegTable[0];          // 1 桁目 ｾｯﾄ 0
        set_imask_ccr(0);  // 割り込みｲﾈｰﾌﾞﾙ
        break;

    // 出力周波数変更 f=1000Hz
    case 7:
        set_imask_ccr(1);  // 割り込みﾃﾞｨｾｰﾌﾞﾙ
        P_TMRZ.TSTR.BIT.STR1 = 0;    // ﾀｲﾏ Z TCNT_1 ｶｳﾝﾄ停止
        P_TMRZ1.TCR1.BYTE = 0x20;    // CLK=X'tal
        P_TMRZ1.GRA1 = 10000;        // GRA 設定
        P_TMRZ.TSTR.BIT.STR1 = 1;    // ﾀｲﾏ Z TCNT_1 ｶｳﾝﾄ開始
        SegBuf[3] = 0;               // 4 桁目 ｾｯﾄ
        SegBuf[2] = 0;               // 3 桁目 ｾｯﾄ
        SegBuf[1] = 0;               // 2 桁目 ｾｯﾄ
        SegBuf[0] = SegTable[1]|0x80;     // 1 桁目 ｾｯﾄ 1.
        set_imask_ccr(0);  // 割り込みｲﾈｰﾌﾞﾙ
        break;
```

```
        // 出力周波数変更 f=2000Hz
        case 8:
             set_imask_ccr(1);  // 割り込みﾃﾞｨｾｰﾌﾞﾙ
             P_TMRZ.TSTR.BIT.STR1 = 0;    // ﾀｲﾏ Z TCNT_1 ｶｳﾝﾄ停止
             P_TMRZ1.TCR1.BYTE = 0x20;    // CLK=X'tal
             P_TMRZ1.GRA1 = 5000;         // GRA 設定
             P_TMRZ.TSTR.BIT.STR1 = 1;    // ﾀｲﾏ Z TCNT_1 ｶｳﾝﾄ開始
             SegBuf[3] = 0;               // 4 桁目 ｾｯﾄ
             SegBuf[2] = 0;               // 3 桁目 ｾｯﾄ
             SegBuf[1] = 0;               // 2 桁目 ｾｯﾄ
             SegBuf[0] = SegTable[2]|0x80;     // 1 桁目 ｾｯﾄ 2.
             set_imask_ccr(0);  // 割り込みｲﾈｰﾌﾞﾙ
             break;

        // 出力周波数変更 f=4000Hz
        case 3:
             set_imask_ccr(1);  // 割り込みﾃﾞｨｾｰﾌﾞﾙ
             P_TMRZ.TSTR.BIT.STR1 = 0;    // ﾀｲﾏ Z TCNT_1 ｶｳﾝﾄ停止
             P_TMRZ1.TCR1.BYTE = 0x20;    // CLK=X'tal
             P_TMRZ1.GRA1 = 2500;         // GRA 設定
             P_TMRZ.TSTR.BIT.STR1 = 1;    // ﾀｲﾏ Z TCNT_1 ｶｳﾝﾄ開始
             SegBuf[3] = 0;               // 4 桁目 ｾｯﾄ
             SegBuf[2] = 0;               // 3 桁目 ｾｯﾄ
             SegBuf[1] = 0;               // 2 桁目 ｾｯﾄ
             SegBuf[0] = SegTable[4]|0x80;     // 1 桁目 ｾｯﾄ 4.
             set_imask_ccr(0);  // 割り込みｲﾈｰﾌﾞﾙ
             break;

        // 出力周波数変更 f=8000Hz
        case 4:
             set_imask_ccr(1);  // 割り込みﾃﾞｨｾｰﾌﾞﾙ
             P_TMRZ.TSTR.BIT.STR1 = 0;    // ﾀｲﾏ Z TCNT_1 ｶｳﾝﾄ停止
             P_TMRZ1.TCR1.BYTE = 0x20;    // CLK=X'tal
             P_TMRZ1.GRA1 = 1250;         // GRA 設定
             P_TMRZ.TSTR.BIT.STR1 = 1;    // ﾀｲﾏ Z TCNT_1 ｶｳﾝﾄ開始
             SegBuf[3] = 0;               // 4 桁目 ｾｯﾄ
             SegBuf[2] = 0;               // 3 桁目 ｾｯﾄ
             SegBuf[1] = 0;               // 2 桁目 ｾｯﾄ
             SegBuf[0] = SegTable[8]|0x80;     // 1 桁目 ｾｯﾄ 8.
             set_imask_ccr(0);  // 割り込みｲﾈｰﾌﾞﾙ
             break;

        // 出力周波数変更 f=16000Hz
        case 5:
             set_imask_ccr(1);  // 割り込みﾃﾞｨｾｰﾌﾞﾙ
             P_TMRZ.TSTR.BIT.STR1 = 0;    // ﾀｲﾏ Z TCNT_1 ｶｳﾝﾄ停止
             P_TMRZ1.TCR1.BYTE = 0x20;    // CLK=X'tal
             P_TMRZ1.GRA1 = 625;     // GRA 設定
             P_TMRZ.TSTR.BIT.STR1 = 1;    // ﾀｲﾏ Z TCNT_1 ｶｳﾝﾄ開始
             SegBuf[3] = 0;               // 4 桁目 ｾｯﾄ
             SegBuf[2] = 0;               // 3 桁目 ｾｯﾄ
             SegBuf[1] = SegTable[1];          // 2 桁目 ｾｯﾄ 1
             SegBuf[0] = SegTable[6]|0x80;     // 1 桁目 ｾｯﾄ 6.
             set_imask_ccr(0);  // 割り込みｲﾈｰﾌﾞﾙ
             break;

        case 0:
             break;
```

```
        case 1:
            break;

        case 2:
            break;
    }
}


/************************************************************
    タイマZ イニシャライズ
************************************************************/
void init_tmrz(void)
{
    P_TMRZ1.TCR1.BYTE  =   0x20;     // CLK=X'tal,GRA ｺﾝﾍﾟｱﾏｯﾁ TCNT ｸﾘｱ
    P_TMRZ1.TIORA1.BYTE=   0x88;     // GRA ｱｳﾄﾌﾟｯﾄｺﾝﾍﾟｱﾚｼﾞｽﾀ,端子出力禁止
    P_TMRZ1.GRA1       =   10000;    // 初期周期設定 / 500μsec
    P_TMRZ1.TIER1.BIT.IMIEA = 1; // TCNT=GRA ｲﾝﾀﾗﾌﾟﾄ ｲﾈｰﾌﾞﾙ
}


/************************************************************
    タイマＢ１ イニシャライズ
************************************************************/
void init_tmrb1(void)
{
    P_TMRB1.TMB1.BYTE  =   0xfb;         // ｵｰﾄﾘﾛｰﾄﾞ,X'tal/256
    P_TMRB1.TCB1       =   256-195;  // 2.496msec
}


/************************************************************
    Ｉ／Ｏポート イニシャライズ
************************************************************/
void init_port(void)
{
    P_PORT.PMR1.BYTE   =   0x00;     // P10,11,14-17,22,72 使用
    P_PORT.PCR1.BYTE   =   0xf0;     // P10-12 In / P14-17 Out
    P_PORT.PUCR1.BYTE  =   0x00;     // ﾌﾟﾙｱｯﾌﾟ MOS 使用しない
    P_PORT.PDR1.BYTE   =   0x00;     // P14-17 = Low

    P_PORT.PCR2.BYTE   =   0x18;     // P20-22 In / P23,24 Out
    P_PORT.PDR2.BYTE   =   0x00;     // P23,24 = Low
    P_PORT.PMR3.BYTE   =   0x00;     // P23,24 CMOS Out

    // P60:DIN  P61:SCLK  P62:/SYNC  P63-67:output of check
    P_PORT.PCR6.BYTE   =   0xFF;     // P60-67 Out
    P_PORT.PDR6.BYTE   =   0x06;     // P61,P62=High / P60,P63-P67=Low

    P_PORT.PCR7.BYTE   =   0x77;     // P70-72,74-76 Out
    P_PORT.PDR7.BYTE   =   0x00;     // P70-72,74-76 = Low
}


/************************************************************
    割込みコントローラ イニシャライズ
************************************************************/
```

```
void init_int(void)
{
    P_INT.IRR2.BIT.IRRTB1   =    0;   // ﾀｲﾏ B1 割込み要求ﾌﾗｸﾞ ｸﾘｱ
    P_INT.IENR2.BIT.IENTB1  =    1;   // ﾀｲﾏ B1 割込み要求ｲﾈｰﾌﾞﾙ
}


/************************************************************
    End of Dynamic Scan Program.
************************************************************/
```

第4章 LAN によるデータ転送

市販されている多くのパソコンに LAN ポートが搭載されるようになり，LAN によるネットワークを組むのが普通になってきました。そこで使われるプロトコルは Windows などの OS によってサポートされており，ユーザはプロトコルの事を殆んど意識することなくネットワークを組むことができます。

パソコンを使用しているだけならそれで十分ですが，マイコンを使って LAN 経由でハードウェアを制御しようとするとあまりにもブラックボックス化されていて手が出ないのが実情ではないでしょうか。

この章では，UDP と呼ばれるプロトコルを利用して，パソコンを使って LAN 経由で‘TK-3687&I/F トレーニングユニット’からデータを取得するプログラムを見ていきます。LAN によるハードウェアの制御の入口となるところです。

なお，もっと詳しく知りたい，という方は，別冊の「I/F トレーニングユニットユーザーズマニュアル<応用編>」をご覧下さい。TCP/IP プロトコルを使ったアプリケーションプログラムが記載されています。

パソコンの設定

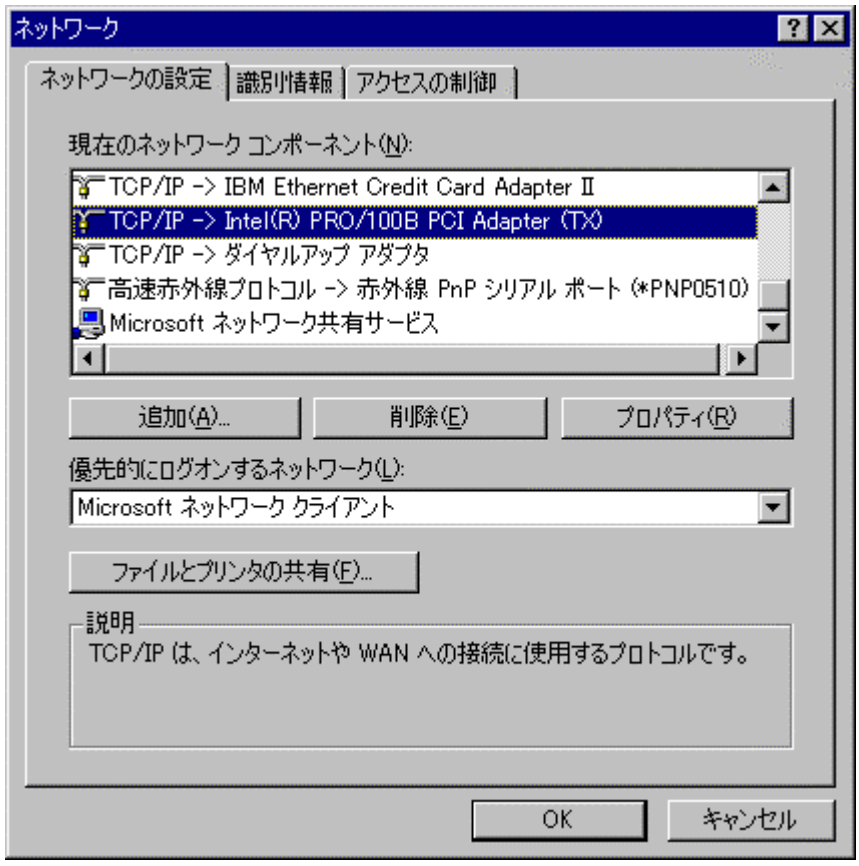
実習の前にパソコンの設定を行います。これから作成するプログラムでは TK-3687 に IP アドレス“192．168．0．200”を割り当てています。Windows などの OS には IP アドレスを自動的に割り当てる機能が組み込まれていますが，それには割り当てを受ける側がその機能に対応している必要があります。ここでは学習という事でその機能は省いていますので，パソコンの設定を変更してパソコンも含む各端末の IP アドレスを固定します。なお，上記の変更を行なうと他の機器と交信できなくなる場合があります。その際は，ネットワーク管理者，または弊社のサポートまでお問い合わせください。

まず，“コントロールパネル”から“ネットワーク”を開きます。

次に“現在のネットワークコンポーネント”から

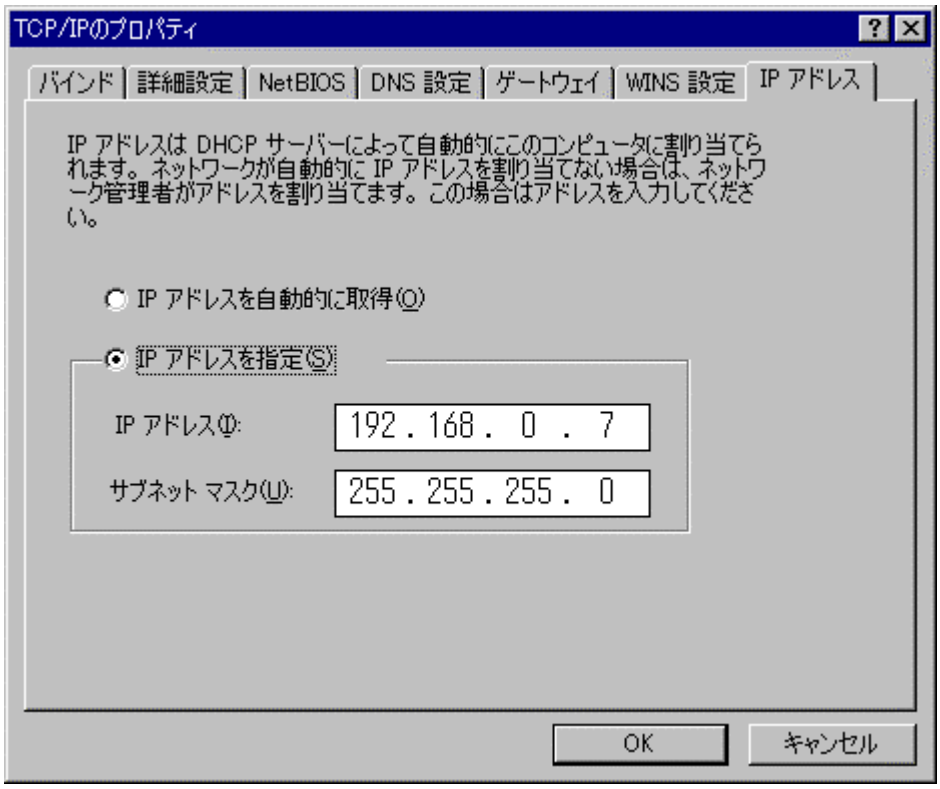
“TCP/IP -> ...”

を選択してプロパティを開きます。（...にはネットワークボード等の名称が入ります。使用しているネットワークアダプタを選択して下さい。）



“IP アドレスタブ”を選んだのち，“IP アドレスを指定”をチェックして，次の値を入力します。

IP アドレス　　192. 168. 0. n※
サブネットマスク　　255. 255. 255. 0
※0～255 の間で 200 を除いた値であればなんでもかまいません。但し，他の IP アドレスとぶつからない様にして下さい。



最後に“OK”を押すと設定を有効にするために再起動を促されるので，再起動します。以上でパソコンの設定は終了です。

パソコンの設定が終了したら TK-3687 をネットワークに接続します。HUB を使っている場合はストレートの LAN ケーブルを使用します。パソコンと直接接続する場合はクロスケーブルを使用します。

◆

これで準備は整いました。それでは，まずサンプルプログラムを動かして動作を確認しておきましょう。

1. サンプルプログラムの実行

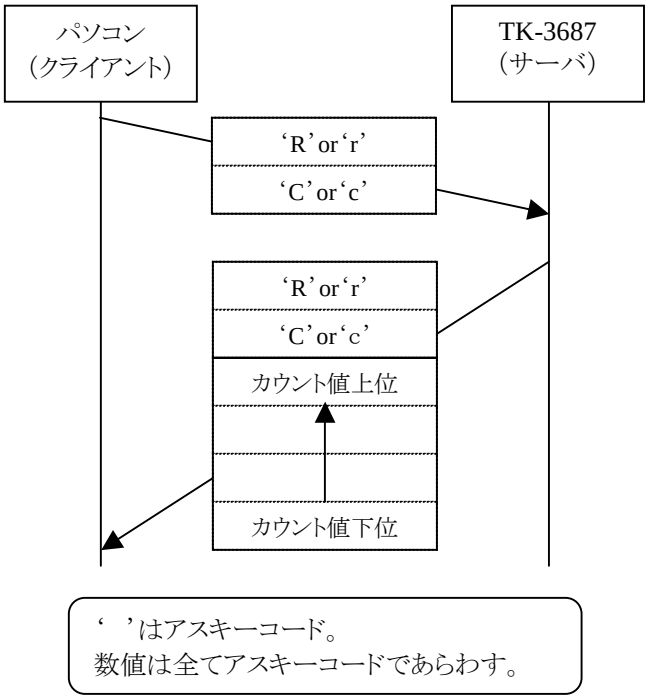
まず TK-3687 で実習用プログラムを実行します。‘FDT’を使って H8/3687 のフラッシュメモリにハイパーH8 と一緒に書き込んでください。7400h 番地から実行すると，I/Fトレーニングユニットの 7 セグメント LED にカウントアップする 16 進数が表示されます。

次に，CD-ROM の‘udp12c.exe’を実行してください。右の画面が表示されます。

‘接続’ボタンをクリックするとデータ送信が始まります。送信モニタの欄に送信データの内容が表示されます。そして，受信データの欄に受信した 7 セグメント LED に表示されている数値データが表示されます。

パソコンと TK-3687 は次のようなやり取りをしています。TK-3687 はパソコンから‘RC’を受信するとその時のカウント値をパソコンへ送信します。

★ カウンタの値のリード ☆



あとはこれをLANのプロトコルに従ってプログラムするだけですが，まず，LANのプロトコルの基本的な考え方からみてみましょう。

2. LAN の概要

LAN で使われているプロトコル（TCP, UDP, IP など）は単独で存在しているのではありません。次の図をご覧下さい。

プロトコル				TCP/IP の 4 層モデル
FTP, Telnet, SMTP HTTP, SSL ．．．	BootP, DHCP TFTP, DNS ．．．			アプリケーション層
TCP	**UDP**			トランスポート層
IP		ARP	ICMP	インターネット層
Ethernet・IEEE802.3(CSMA/CD) LAN ．．．				データ・リンク層

図 4-1 ネットワークの 4 層モデル

今回使用するプロトコルは UDP です。図 4-1 からわかるように, UDP の下位プロトコルは IP, その下位は Ethernet フレームです。では, 実際に送受信したいデータと, これらのプロトコルはどのような関係にあるのでしょうか。

キーワードはカプセル化です。図 4-2 をご覧下さい。UDP で何かのメッセージ（例えば‘RC’というテキストデータ）を送りたいとします。まずはそのメッセージに UDP の情報（ヘッダ）を付加します。同じように UDP の下位プロトコルは IP なので, さらに IP の情報（ヘッダ）を付加します。最後に Ethernet フレームのヘッダとフレームチェックシーケンス（FCS）を付加して完成です。このようにプロトコルごとに情報を付加していくことをカプセル化と呼びます。データの送り手は次々にカプセル化を繰り返して Ethernet フレームにしてから送信します。

データの受け手はこれとは逆にカプセルを次々に外してメッセージを取り出します。

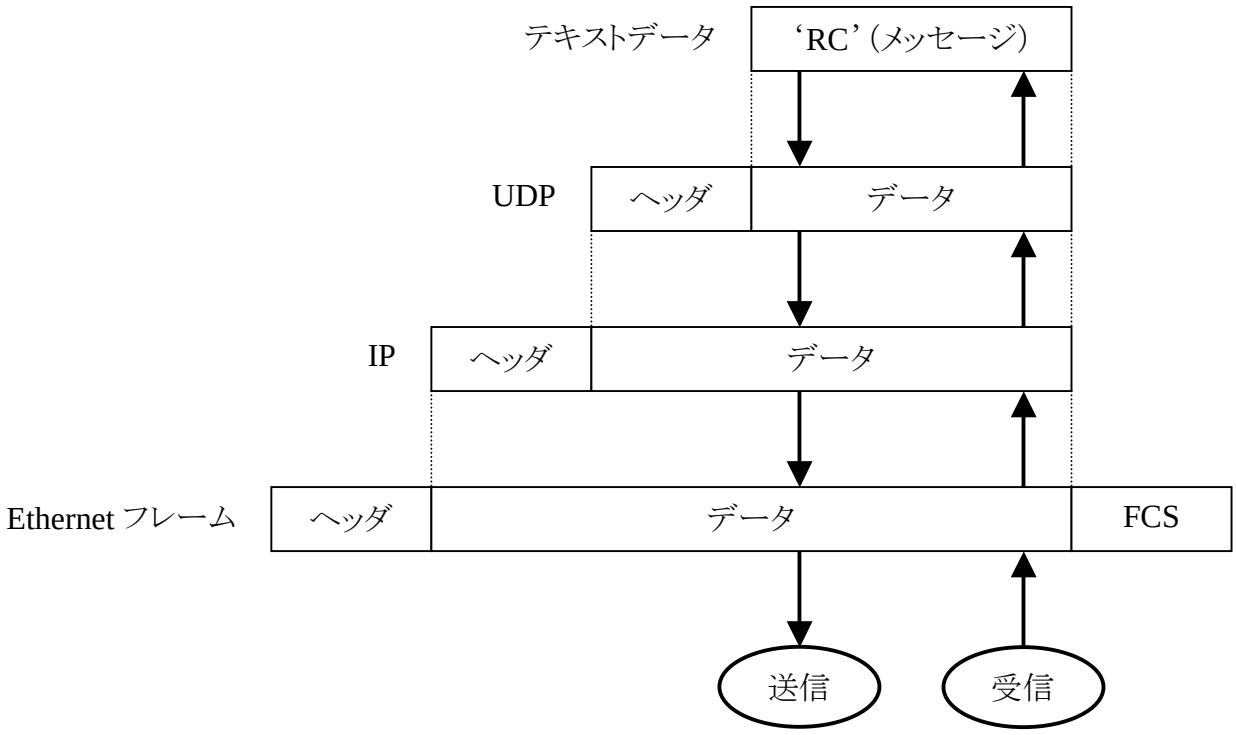


図 4-2 データはプロトコルごとに情報を付加され階層をなす

それでは最も下位層になる Ethernet フレームの受信ルーチンをみてみましょう。LAN データの受信はまずここから始まります。プログラムもここから作成します

3. Ethernet フレームの受信

Ethernet フレームは次のような構成をしています。

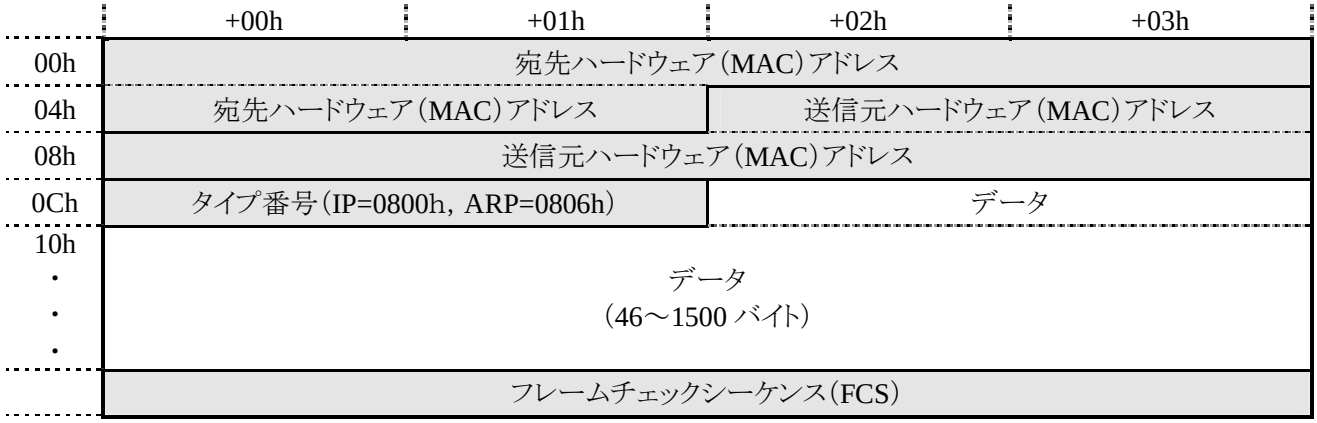


図 4-3 Ethernet フレームの構成

Ethernet フレームの受信は，‘receive_ethernet_frame()’関数で行なっています。コールすると，受信データがあるか判定し，あった場合は受信データ全てを H8/3687 内の RAM へ読み込みます。読み込んだ受信データからプロトコルが何かを判定し，プロトコルナンバーを決定します。関数の戻り値はプロトコルナンバーです。

受信データがなかった場合や，判定できないプロトコルの場合，関数の戻り値はエラーナンバーです。

‘receive_ethernet_frame()’関数のソースリストは次の通りです。

```
/*************************************************************************
    イーサネットフレームの受信
-------------------------------------------------------------------------
    戻り値    受信OKのときはプロトコルナンバー
                  10h:ARP
                  20h:TCP
                  21h:UDP
                  22h:ICMP
              受信NGのときはエラーナンバー
                   -1:ISRのRXE,TXE,OVWエラー
                   -2:受信データなし
                   -3:パケット異常受信(RSR,bit0)
                   -4:イーサネットタイプサポート外
                   -5:イーサネットタイプサポート外(未使用)
                   -6:IPバージョンサポート外
                   -7:IPプロトコルサポート外
                   -8:受信フレームバイト数オーバー
*************************************************************************/
int receive_ethernet_frame(void)
{
    int  r = 0;
    unsigned char _bnry,_curr;

    //ページ0にセット
    write_rtl8019as(0x22,CR);

    //ISRのRXE,TXE,OVWをチェック
    if ((read_rtl8019as(ISR)&0x1c)!=0){
        init_rtl8019as();  //エラー時は再イニシャライズ
        return -1;
    }
```

```
    //受信パケットがあるかチェック
    write_rtl8019as(0x62,CR);    //ページ1
    _curr = read_rtl8019as(CURR);

    write_rtl8019as(0x22,CR);    //ページ0
    _bnry = read_rtl8019as(BNRY) + 1;
    if (_bnry>=PAGE_STOP)   _bnry = PAGE_START;

    if (_bnry==_curr)  return -2;     //受信していない

    //受信フレームの情報を得る（リングバッファの先頭4バイト）
    read_remote_dma(_bnry*0x100,4,&EthernetFrame.RSR_COPY);

    //RTL8019ASからのRBCは上位と下位がH8と逆になっているのでここで調整する
    EthernetFrame.RBC = (EthernetFrame.RBC & 0x00ff) * 0x0100
                      + (EthernetFrame.RBC>>8);

    //パケットの受信状況をチェック
    if ((EthernetFrame.RSR_COPY&0x01)==0) {r = -3;} //パケット異常受信

    //受信フレームのバイト数チェック
    else if (EthernetFrame.RBC>0x0600)         {r = -8;} //多すぎる

    else{
        //リモートDMAでフレーム本体を取り込む
        read_remote_dma(_bnry*0x100+4,EthernetFrame.RBC,&EthernetFrame.RSR_COPY+4);

        //イーサネットタイプのチェック
        if (EthernetFrame.TYPE==0x0806)        {r = 0x10;}    //ARP
        else if (EthernetFrame.TYPE==0x800){                  //IP
            //IPバージョンのチェック
            if (EthernetFrame.DATA.IP.VER_IHL==0x45){            //IPバージョン→IP
                //IPプロトコルチェック
                if      (EthernetFrame.DATA.IP.PROT==6)        {r = 0x20;}    //TCP
                else if (EthernetFrame.DATA.IP.PROT==17)       {r = 0x21;}    //UDP
                else if (EthernetFrame.DATA.IP.PROT==1)        {r = 0x22;}    //ICMP
                else {r = -7;} //IPプロトコルサポート外
            }
            else {r = -6;} //IPバージョンサポート外
        }
        else {r = -4;} //イーサネットタイプサポート外
    }

    //BNRY更新
    if ((EthernetFrame.NPP<PAGE_START)||(EthernetFrame.NPP>=PAGE_STOP)){
        init_rtl8019as();  //エラー時は再イニシャライズ
        return r;
    }
    write_rtl8019as(0x22,CR);
    write_rtl8019as(EthernetFrame.NPP-1,BNRY);

    return r;
}
```

取り込んだ Ethernet フレームは H8/3687 のメモリ上にセットされますが，プロトコルの階層構造によって同じアドレスが違う意味を持ちます。それで，構造体と共用体を使って Ethernet フレームをあらわします。

```
// イーサネットフレームに関する変数 ----------------------------------
#pragma pack 1
struct ICMP_ST{                         //ICMP構造体
    unsigned char TYPE;                 // タイプ
    unsigned char CODE;                 // コード
    unsigned int  CS;                   // チェックサム
    unsigned int  ID;                   // 識別子
    unsigned int  SEQ;                  // シーケンス番号
    unsigned char DATA[1490];           // データ
};

#pragma pack 1
struct UDP_ST{                          //UDP構造体
    unsigned int  SRC;                  // 送信元ポート
    unsigned int  DEST;                 // 宛先ポート
    unsigned int  L;                    // セグメント長
    unsigned int  CS;                   // チェックサム
    unsigned char DATA[1490];           // データ
};

#pragma pack 1
struct TCP_ST{                          //TCP構造体
    unsigned int  SRC;                  // 送信元ポート
    unsigned int  DEST;                 // 宛先ポート
    unsigned long SEQ;                  // シーケンス番号
    unsigned long ACK;                  // 応答確認番号
    unsigned char DO;                   // データオフセット(上位4ビット)
    unsigned char CF;                   // コントロールフラグ(下位6ビット)
    unsigned int  WINDOW;               // ウィンドウ
    unsigned int  CS;                   // チェックサム
    unsigned int  URGPTR;               // 緊急ポインタ
    unsigned char DATA[1478];           // データ
};

#pragma pack 1
union IP_DATA_UNI{                      //IPデータ共用体
    struct ICMP_ST ICMP;                // ICMPのときのIPデータ
    struct UDP_ST  UDP;                 // UDPのときのIPデータ
    struct TCP_ST  TCP;                 // TCPのときのIPデータ
    unsigned char DATA[1498];           // 汎用
};

#pragma pack 1
struct IP_ST{                           //IP構造体
    unsigned char VER_IHL;              // バージョンとヘッダ長
    unsigned char TOS;                  // サービスタイプ
    unsigned int  TL;                   // パケット長
    unsigned int  ID;                   // 識別子
    unsigned int  FO;                   // フラグ(上位3ビット)と
                                        //   フラグメントオフセット(下位13ビット)
    unsigned char TTL;                  // 生存時間
    unsigned char PROT;                 // プロトコル
    unsigned int  HC;                   // ヘッダチェックサム
```

```
    unsigned char  SOURCE[4];         // 送信元IPアドレス
    unsigned char  DEST[4];           // 宛先IPアドレス
    union IP_DATA_UNI  DATA;          // IPデータ
};

#pragma pack 1
struct ARP_ST{                        //ARP構造体
    unsigned int   TYPE;              // ハードウェアタイプ
    unsigned int   PROT;              // プロトコルタイプ
    unsigned char  HL;                // ハードウェアアドレス長
    unsigned char  PL;                // プロトコルアドレス長
    unsigned int   OPE;               // オペレーションコード
    unsigned char  SOURCE_MAC[6];     // 送信元ハードウェア(MAC)アドレス
    unsigned char  SOURCE_IP[4];      // 送信元プロトコル(IP)アドレス
    unsigned char  DEST_MAC[6];       // ターゲットハードウェア(MAC)アドレス
    unsigned char  DEST_IP[4];        // ターゲットプロトコル(IP)アドレス
    unsigned char  DATA[1490];        // データ
};

#pragma pack 1
union ETHER_DATA_UNI{                 //Ethernetデータ共用体
    struct IP_ST   IP;                // IPのときのデータ
    struct ARP_ST  ARP;               // ARPのときのデータ
    unsigned char  DATA[1518];        // 汎用
};

#pragma pack 1
struct ETHERNET_FRAME_ST{             //Ethernetフレーム構造体
    unsigned char  RSR_COPY;          // RTL8019AS RSRレジスタのコピー
    unsigned char  NPP;               // RTL8019AS 次のフレームのページ番号
    unsigned int   RBC;               // RTL8019AS 受信フレームのバイト数
    unsigned char  DEST_MAC[6];       // 宛先MACアドレス
    unsigned char  SOURCE_MAC[6];     // 送信元MACアドレス
    unsigned int   TYPE;              // タイプ
    union ETHER_DATA_UNI    DATA;     // データ
};

struct ETHERNET_FRAME_ST     EthernetFrame; //Ethernetフレーム
```

例えば, UDP プロトコルの送信元ポート番号を知りたい時は,

EthernetFrame.DATA.IP.DATA.UDP.SRC

をアクセスします。

さて, これで Ethernet フレームを取り込むことができました。次はさらに上位層をチェックしていきます。なお, プログラムをみればわかるとおり Ethernet フレームのタイプが IP のときは, さらに一つ上位層の IP プロトコルのタイプもチェックして UDP を受信したかどうかも判定しています。というわけで, 次は UDP の送受信です。

4. UDP の構成と送受信

UDP は IP パケットに載せて送られてきます。IP パケットのヘッダを付加した形で UDP パケットの構成を示します。

	+00h		+01h	+02h	+03h
IP ヘッダ	バージョン	ヘッダ長	サービスタイプ	パケット長	
	識別子			フラグ	フラグメントオフセット
	生存時間		プロトコル№. ICMP=1, TCP=6 UDP=17	ヘッダチェックサム	
	送信元IPアドレス				
	宛先IPアドレス				
	オプション（ないこともある）				
UDP ヘッダ	送信元ポート			宛先ポート	
	セグメント長			チェックサム	
	データ（可変長）				

図 4-4 IP/UDP のデータ構成

UDP を受信したら，次は‘udp_receive()’関数をコールします。この中で次の判定をしていきます。

① 自分宛かどうか IP アドレスをチェックする。
② ポートアドレスが 10001 かチェックする。
③ コマンドが‘RC’かチェックする。

ソースリストは次の通りです。

```
/*****************************************************************************
    UDP 受信
*****************************************************************************/
void udp_receive(void)
{
    unsigned int i;
    unsigned char cmd[2];

    //自分宛てか確認，IP アドレスを比較する
    for (i=0; i<4; i++){
        if (EthernetFrame.DATA.IP.DEST[i]!=MyIpAddress[i])   return;
    }

    //コマンドをゲット
    cmd[0] = EthernetFrame.DATA.IP.DATA.UDP.DATA[0];
    cmd[1] = EthernetFrame.DATA.IP.DATA.UDP.DATA[1];

    //宛先ポートによって処理を変更する
    switch (EthernetFrame.DATA.IP.DATA.UDP.DEST){
        case 10001:         //---------------------------------------------
```

```
            switch (cmd[0]){
                case 'R': //リード
                case 'r':
                    switch (cmd[1]){
                        case 'C': //リードカウンタ
                        case 'c':
                            get_count();  break;
                    }
                    break;
            }
            break;
    }
}
```

コマンドが正しければ表示されているカウント値を送信します。‘get_count()’関数のソースリストは次の通りです。

```
/*************************************************************************
    表示値を送信
*************************************************************************/
void get_count(void){
    EthernetFrame.DATA.IP.DATA.UDP.DATA[2] = cnv_hex2asc(DispBuf[3]);
    EthernetFrame.DATA.IP.DATA.UDP.DATA[3] = cnv_hex2asc(DispBuf[2]);
    EthernetFrame.DATA.IP.DATA.UDP.DATA[4] = cnv_hex2asc(DispBuf[1]);
    EthernetFrame.DATA.IP.DATA.UDP.DATA[5] = cnv_hex2asc(DispBuf[0]);
    udp_send(6);
}
```

最後にUDPで送信します。Ethernetフレームには受信したUDPのヘッダやIPのヘッダがセットされたままです。これをもとに，宛先MACアドレスや宛先IPアドレス，宛先ポートなどをセットします。‘send_udp()’のソースリストは次の通りです。

```
/*************************************************************************
    UDP 送信
-------------------------------------------------------------------------
    引数 length         送信データバイト数
*************************************************************************/
void udp_send(unsigned int length)
{
    unsigned int i,l,port;

    //イーサネットフレームのセット
    for (i=0; i<6; i++){    //宛先 MAC アドレスセット
        EthernetFrame.DEST_MAC[i] = EthernetFrame.SOURCE_MAC[i];
    }
    for (i=0; i<6; i++){    //送信元 MAC アドレスセット
        EthernetFrame.SOURCE_MAC[i] = MyMacAddress[i];
    }
    for (i=0; i<4; i++){    //宛先 IP アドレスセット
        EthernetFrame.DATA.IP.DEST[i] = EthernetFrame.DATA.IP.SOURCE[i];
    }
    for (i=0; i<4; i++){    //送信元 IP アドレスセット
        EthernetFrame.DATA.IP.SOURCE[i] = MyIpAddress[i];
    }
    port = EthernetFrame.DATA.IP.DATA.UDP.DEST;
    EthernetFrame.DATA.IP.DATA.UDP.DEST             //宛先ポートセット
        = EthernetFrame.DATA.IP.DATA.UDP.SRC;
```

```
    EthernetFrame.DATA.IP.DATA.UDP.SRC = port; //送信元ポートセット
    EthernetFrame.DATA.IP.DATA.UDP.L = length + 8;  //UDP セグメント長セット
    EthernetFrame.DATA.IP.DATA.UDP.CS = 0x0000;//UDP チェックサムクリア（使用しない）
    EthernetFrame.DATA.IP.TL = length + 8 + 20;//IP ヘッダパケット長セット
    EthernetFrame.DATA.IP.HC = 0x0000;     //IP ヘッダチェックサムクリア
    EthernetFrame.DATA.IP.HC     //IP ヘッダチェックサムセット
        = cal_checksum(&EthernetFrame.DATA.IP.VER_IHL,20,0,1);    //IP データグラムのヘッダ部分

    //イーサネットフレームの総バイト数を計算
    l = length + 8 + 20 + 14;     //UDP データ+UDP ヘッダ+IP ヘッダ+イーサネットフレームヘッダ
    if (l<60) l=60;     //60 バイト未満のときは 60 にする

    //イーサネットフレームの送信
    transmit_ethernet_frame(l);
}
```

◆

以上で TK-3687 の LAN の説明は終わりです。もちろん，これはほんの触りに過ぎません。例えば，

① 相手の MAC アドレスはどうしたらわかるのか。

② インターネットで使われている TCP というプロトコルはどのようなものか。

③ web ブラウザに表示するにはどうしたらよいのか。

④ LAN コントローラ，RTL8019AS の使い方は。

など，興味は尽きないことでしょう。このテキストで説明するには紙面が足りません。それで，LAN についてさらに興味のある方は，「I/F トレーニングユニットユーザーズマニュアル<応用編>」をぜひお読みください。現在<応用編>には下記のものが揃っています。

◆

マイコンサーバシステム

パソコンのブラウザから I/F トレーニングユニットのハードウェアを制御し値を取得する方法を学習します。主に TCP による通信を学習します。

LAN 電話（LAN による音声通話システム）

I/F トレーニングユニット同士で LAN を介して音声通話をする方法を考えます。主に UDP による通信を学習します。

◆

これらのマニュアルを読むことでマイコンによる LAN 制御についての理解がさらに深まることと思います。

最後に，パソコンのプログラムについて取り上げます。

5. VisualBasic のソースリスト

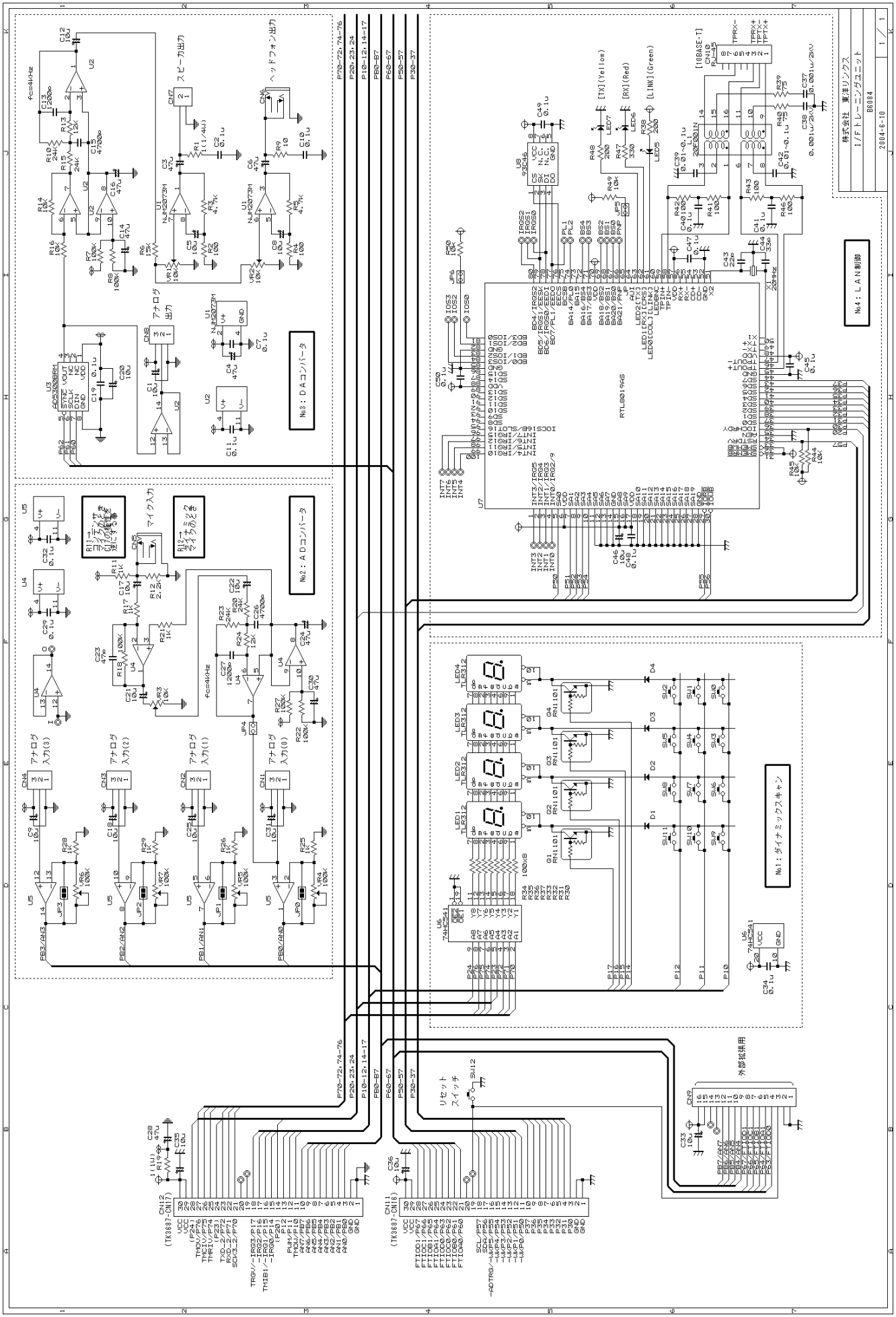
以下にパソコン側のソフトのソースリストを示します。‘RC’コマンドの内容と対応させれば各プロシージャで何を行なっているのか大体の見当はつくのではないでしょうか。プロパティ等についてはプロジェクトを開いて確認してみて下さい。プログラム名称は‘udp12c. exe’です。プロジェクトファイルとともに付属の CD-ROM に在中されています。

```
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'       "UDP12c.VBP"                                                '
'                                                                   '
'       UDPﾌﾟﾛﾄｺﾙによるTK-3687制御ﾌﾟﾛｸﾞﾗﾑ;                            '
'                                                                   '
'                                                                   '
'       動作環境：       TK-3687                                    '
'       Url:             下記プログラム中で設定可                   '
'                        (defalt：192.168.0.200)                    '
'       control:         MS winsock Control 6.0                     '
'                        MS Windows Common Control 6.0              '
'       protocol:        sckUDPProtocol                             '
'       copyright        2004.4.2/toyolinx/tMogi                    '
'                                                                   '
'                                                                   '
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'   履歴                                                            '
'   20040427/カウント値リードコマンド追加(RC)。                     '
'           /                                                       '
'                                                                   '
'                                                                   '
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

Option Explicit

Dim cnt As Byte
Dim i As Integer, j As Integer, k As Integer
Dim m As String, n(20) As String
Dim x As Variant, y As Variant, z As Variant

Dim recvtimes As Integer                        '受信連番
Dim Rxdata As String                            '受信データ
Dim Txdata As String                            '送信データ

'-----------------------------------------------------------------
Private Sub form_load()                         '
    tinput.Text = ""
    thost.Text = "192.168.0.200"
End Sub

'-----------------------------------------------------------------
'プログラムの終了
Private Sub close_click()
    Winsock1.close              '回線の切断
    Unload Me
End Sub
```

```
'---------------------------------------------------------------------
'WINsockによるUDP接続. >>> ARP要求、応答により回線接続
Private Sub bind_click()
    On Error GoTo err_handle

    bind.Enabled = False
retry:
    Winsock1.close
    Winsock1.Protocol = sckUDPProtocol          'select UDP
    Winsock1.RemoteHost = thost.Text
    Winsock1.RemotePort = 10001
    Winsock1.bind 1002

'カウント値送信要求
        recvtimes = 0
        Txdata = "RC"           'カウント値要求
        Winsock1.SendData Txdata
        tinput = Txdata & "/" & msg(Txdata)

        Timer1.Enabled = True       'timer-on
        Timer1.Interval = 1000      'unit:msec
    Exit Sub

err_handle:
        tinput.Text = "接続不良"
        GoTo retry
End Sub

'---------------------------------------------------------------------
'1秒タイマイベント発生
Private Sub timer1_timer()
    Beep
    cnt = cnt + 1
    Txdata = "RC"               'カウント値要求
    Winsock1.SendData Txdata
    tinput = Txdata & "/" & msg(Txdata)

End Sub

'---------------------------------------------------------------------
'データ受信イベント
Private Sub winsock1_dataarrival(ByVal bytestotal As Long)
    recvtimes = recvtimes + 1
    Winsock1.GetData Rxdata
    Call scroll(recvtimes, Rxdata)
End Sub

'---------------------------------------------------------------------
Private Function scroll(x As Integer, y As String)
    z = "" + msg(y)
    'z = "" + y
    j = 20
    If LB.ListCount <= j Then GoTo scroll_01
    For i = 1 To j
        LB.List(i - 1) = LB.List(i)
    Next i
    LB.AddItem Format(CStr(x), "000  ") & Rxdata & "/" & z, j
    LB.RemoveItem j + 1
```

```
    GoTo scroll_02
scroll_01:
    LB.AddItem Format(CStr(x), "000  ") & Rxdata & "/" & z
scroll_02:
End Function

'-----------------------------------------------------------------------
Private Function msg(s As String) As String
    j = Len(s)
    m = ""
    For k = 1 To j
        x = Mid(s, k, 1)
        y = Asc(x)
        n(k) = Hex(y)
        Select Case y
          Case 0 To 15
            n(0) = " 0"
          Case Else
            n(0) = " "
        End Select
        m = m & n(0) & n(k)
    Next
   msg = m
End Function

'-----------------------------------------------------------------------
Private Sub winsock_error(ByVal number As Integer, _
                          description As String, _
                          ByVal scode As Long, _
                          ByVal source As String, _
                          ByVal helpfile As String, _
                          ByVal helpcontext As Long, _
                          canceldisplay As Boolean)
    SB.SimpleText = description
End Sub

'-----------------------------------------------------------------------
'受信データクリア
Private Sub clear_Click()
    Do Until LB.ListCount = 0
        LB.RemoveItem (0)
    Loop
End Sub
```

付録



パーツリスト(1/2)

	部品番号	型名，規格	メーカー	数量	備考
1	U1	NJM2073M	新日本無線	1	
2	U2,U4,U5	MC33204D	オンセミ	3	入出力レイルツーレイル，相当品可，U4 は未実装
3	U3	AD5300BRM	アナデバ	1	
4	U6	74HC541AF		1	フラットパッケージ
5	U7	RTL8019AS	REALTEK	1	
6	U8	M93C46-WMN AT93C46-10SI-2.7	ST マイクロ ATMEL	1	相当品可
7					
8	Q1,2,3,4	RN1101	東芝	4	抵抗内蔵トランジスタ
9	D1,2,3,4	1S1588		4	相当品可
10	LED1,2,3,4	TLR312	東芝	4	
11	LED5	(緑/LINK)		1	表面実装も使用可能
12	LED6	(赤/RX)		1	表面実装も使用可能
13	LED7	(黄/TX)		1	表面実装も使用可能
14					
15	R1	1Ω(1/4W)		1	
16	R19	1Ω(1W)		1	
17	R9	10Ω		1	未実装
18	R39,40	75Ω		2	
19	R2,4,30,31,32,33,34 R35,36,37,41,42,43 R46	100Ω		14	R4 は未実装
20	R38	200Ω(緑用/LINK)		1	LED5 にあわせて抵抗値調整
21	R47	330Ω(赤用/RX)		1	LED6 にあわせて抵抗値調整
22	R48	200Ω(黄用/TX)		1	LED7 にあわせて抵抗値調整
23	R11,17,21,25,26,28 R29	1KΩ		7	R11,17,21 は未実装
24	R12	2.2KΩ		1	未実装
25	R3,5	4.7KΩ		2	R5 は未実装
26	R14,16,44,45,49,50	10KΩ		6	R49,50 は未実装
27	R6	15KΩ		1	
28	R13,24	12KΩ		2	R24 未実装
29	R10,15,20,23	24KΩ		4	R20,23 は未実装
30	R7,8,18,22,27	100KΩ		5	R18,22,27 は未実装
31					
32	C43	22pF		1	X1 がコンデンサ内蔵型のときは未実装
33	C44	33pF		1	X1 がコンデンサ内蔵型のときは未実装
34	C23	47pF		1	未実装
35	C39,42	0.01μF～0.1μF		2	
36	C2,7,10,11,19,29,32 C34,40,41,45,47,48 C49,50	0.1μF		15	
37	C37,38	DEBF33D102ZC1B 0.001μF/2KV	ムラタ	2	相当品可
38	C13,27	1200pF(フィルム)		2	C27 は未実装
39	C15,26	4700pF(フィルム)		2	C26 は未実装

パーツリスト(2/2)

	部品番号	型名，規格	メーカー	数量	備考
40	C1,5,8,9,12,17,18 C20,21,22,25,31 C33,35,36,46	10μF/16V(電解)		16	C17,21,22,33 は未実装
41	C3,4,6,14,16,24 C28,30	47μF/16V(電解)		8	C6,24,30 は未実装
42					
43	VR1,2,3	CT-6P/10KΩ	コパル	3	VR2,3 は未実装，相当品可
44	VR4,5,6,7	CT-6P/100KΩ	コパル	4	相当品可
45					
46	X1	20MHz		1	コンデンサ内蔵タイプ実装可
47					
48	L1	20F001N	YCL	1	10Base-T Low Pass Filter
49					
50	SW0～11	DP1-120	フジソク	12	
51	SW13	SKHHAK	ALPS	1	相当品可
52					
53	CN1,2,3,4,8	B3P-SHF-1AA	JST	5	
54	CN4,6	MJ-354W	マルシン	2	未実装
55	CN7	B2P-SHF-1AA	JST	1	
56	CN9	HIF3FC-16PA-2.54DSA	HRS	1	未実装，相当品可
57	CN10	TM5RJ3-88	HRS	1	
58	CN11,12	HIF3FB-30DA-2.54DSA	HRS	2	裏付け，相当品可
59					
60	PCB	B6084	東洋リンクス	1	
61					

■ 参考文献

トランジスタ技術　2001 年 1 月号
「特集　21 世紀はネットで I/O!」
CQ 出版社

TCP/IP 解析とプログラミング
澤川渡　綱島明浩　共著　オーム社開発局

RTL8019AS データシート
Realtek　Semoconductor 社　http://www.realtek.com.tw/

■ 掲載された回路・プログラム等を利用した結果，生じたトラブルについて弊社は責任を負いかねますのであらかじめご了承ください。

■ プログラム名・システム名・CPU名等，固有名詞は一般に各メーカの商標もしくは，登録商標です。

■ 本書の内容は将来予告なしに変更することがあります。
（2004年6月作成）

株式会社東洋リンクス

※ご質問はメール，または FAX で…
ユーザーサポート係（月～金 10:00～17:00，土日祝は除く）
〒102-0093　東京都千代田区平河町 1-2-2　朝日ビル
TEL:03-3234-0559
FAX:03-3234-0549
E-mail:toyolinx@va.u-netsurf.jp
URL:http://www2.u-netsurf.ne.jp/~toyolinx

20040616