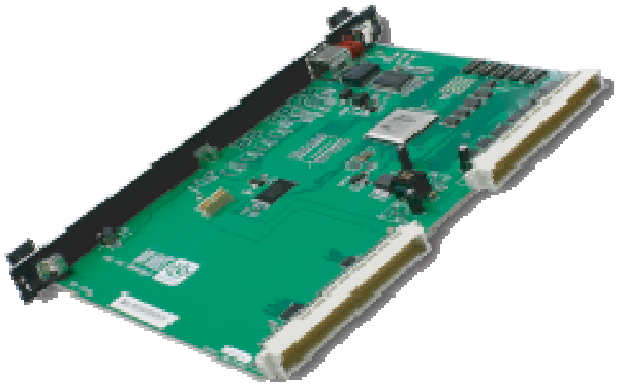
SITCP VME-MASTER MODULE
取扱説明書

Rev 1.1 (Jan. 25, 2010)
Tomohisa Uchida



変更履歴

Rev	変更日	変更ページ	変更内容
0.4	2008 年 2 月 13 日	P12	Address Fix モード時の制限事項を追加
0.5	2008 年 2 月 14 日	P3, 11	非整列転送の非サポートを明記
1.0	2008/04/04	P6	IP Address, TCP port 番号についての説明を追加
1.1	2010/01/25		ボード Rev.3 対応、誤記修正

適用

- ボード Rev.3 に適用する

特徴

- Ethernet を経由した VME slave module 制御
- VME32 マスタ機能サポート

注意

- 現在のバージョンでは複数マスタシステムに対応していません。従って、本モジュールを必ずシステムモジュール(Slot 0)に挿入して使用し、他のマスタ・モジュールを挿入しないで下さい。
- VME 割り込みについてはサポートしていません

インタフェース

以下のインタフェースを持っています。

1. VME インタフェース
2. Ethernet
3. 8 つの汎用 NIM 入力（現バージョンでは未使用）
4. 4 つの汎用 NIM 出力（現バージョンでは未使用）

VME インタフェース

以下の VME マスタ機能をサポートしています。

- D32, D16, D8
- A24, A16
- BLT

データバスの非整列転送はサポートしていません。

A32 はサポートしていません。基板改版後にサポート予定。

NIM 制御入力

コネクタ: LEMO 50Ω終端付

閾値:-360mV (閾値以下で Low level)

無接続時に High レベル

概要

本モジュールは Ethernet 経由で VME バスを制御する為の VME master module です。Ethernet を持つ PC などを使用して Ethernet を経由して本モジュールと同一クレートに挿入されている VME slave module をアクセスする事ができます。

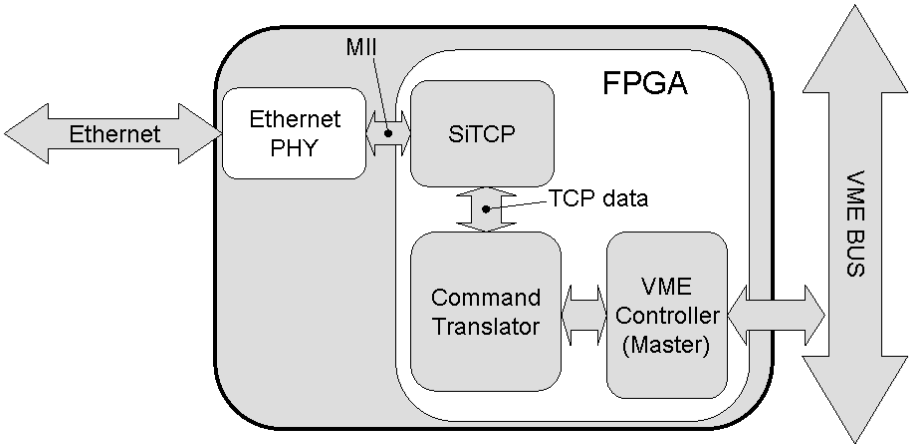


図 1　全体ブロック図

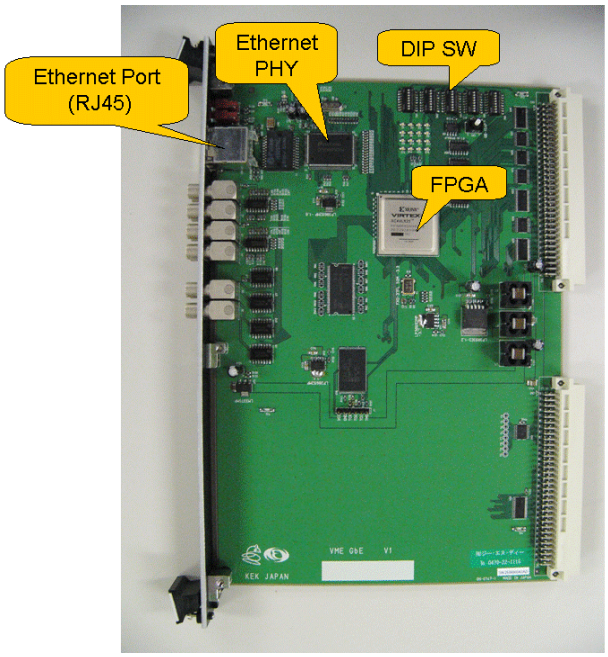


図2　全体写真

図 1 に全体ブロック図、図 2 に全体写真を示します。本モジュールの主要部品は一つの FPGA と Ethernet PHY device です。FPGA と VME バックプレーン間の信号はバスバッファを介して接続されています。

モジュールの設定

電源を入れる前に以下を設定する必要があります。

- IPv4 address
  - モジュールの IP アドレスを設定してください
  - ROM 格納 IP アドレスを使用する時は 0.0.0.0 を設定してください
- TCP Port Number
  - データ通信に使う TCP 待ちうけポートの値を設定してください。
  - ROM 格納 TCP ポート番号を使用する時は 0 を設定してください
- デフォルト値を使用する時
  - ROM データ書き込み等でデフォルト値を使用する場合 IPv4 address=255.255.255.255 を設定してください。下のデフォルト値で動作するようになります。
    - IP address デフォルト値=192.168.10.16
    - TCP port 番号デフォルト値=24

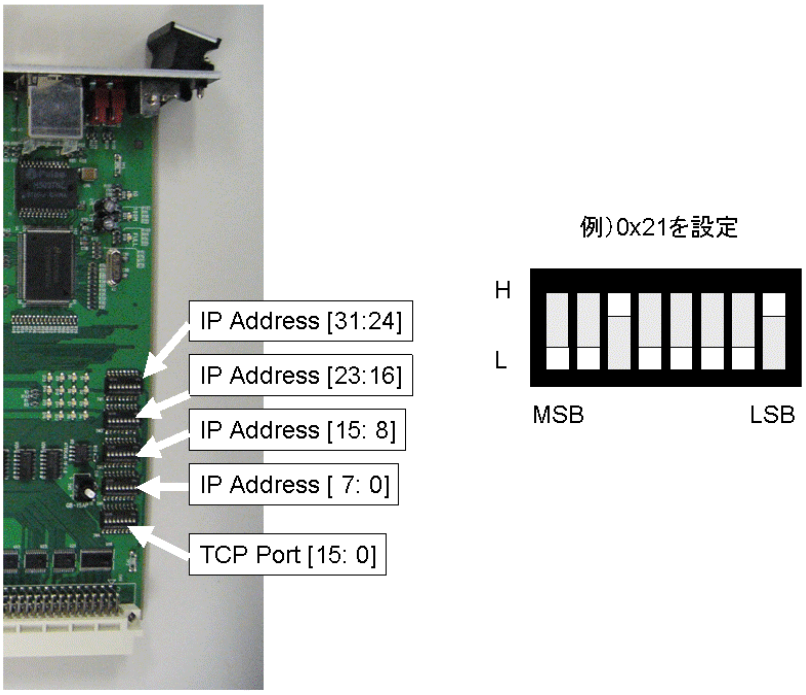


図3に DIP SW の場所と設定例を示します。DIP に 16 進で設定してください。

注意） MAC アドレスは EEPROM に格納されています。MAC アドレスを変更すると動作しなくなるので書き換えないで下さい。設定されている MAC アドレスは基板上に記述してあります。

EEPROM の内容

MAC アドレスや TCP パラメータは EEPROM 内に格納されています。
FPGA 起動時に各パラメータを読み込んで動作を始めます。正しい値を EEPROM に格納しておかないとモジュールが起動しなくなります。

（注意）MAC アドレス
MAC アドレスを変更すると起動しなくなるので注意してください。正常に起動する為にはプロテクトコードを含む ROM 設定ファイルをイーサネットを使用して専用ツールで書き込んでください。モジュールが正常に動作していないときはデフォルト値を読み込ませイーサネット通信ができる状態で書き込みを始めてください。

VME アクセスの概要

イーサネット経由で TCP パケットを交換する事で VME アクセスを行います。

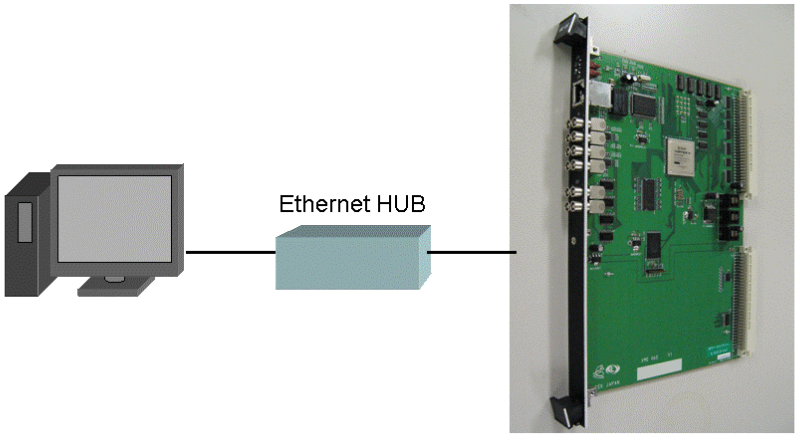


図4　標準システム構成

図4に標準的なシステムを示します。基本的なシステム構成での構成装置は、PC、イーサネット・ハブ、そして本モジュールです。PC は VME バスを制御する為、イーサネット・ハブは PC と本モジュールを接続する為に使用します。

TCP パケットを交換する事で VME アクセスを行うので VME をアクセスするスピードはモジュール性能だけでなくネットワーク性能にも依存します。ネットワークに依存しない安定した性能を求める場合は PC に専用ポートを設け、インターネットなどの公共ネットワークと分離してください。専用ネットワークを使用する場合、性能は PC 性能とアクセスするモジュール数により決まります。

モジュールは複数バスマスタに対応していません。必ず本モジュールはシステム・スロット（Slot 0、通常一番左側のスロット）に挿入してください。また、他のマスタ・モジュールを同一クレート内に挿入しないで下さい。

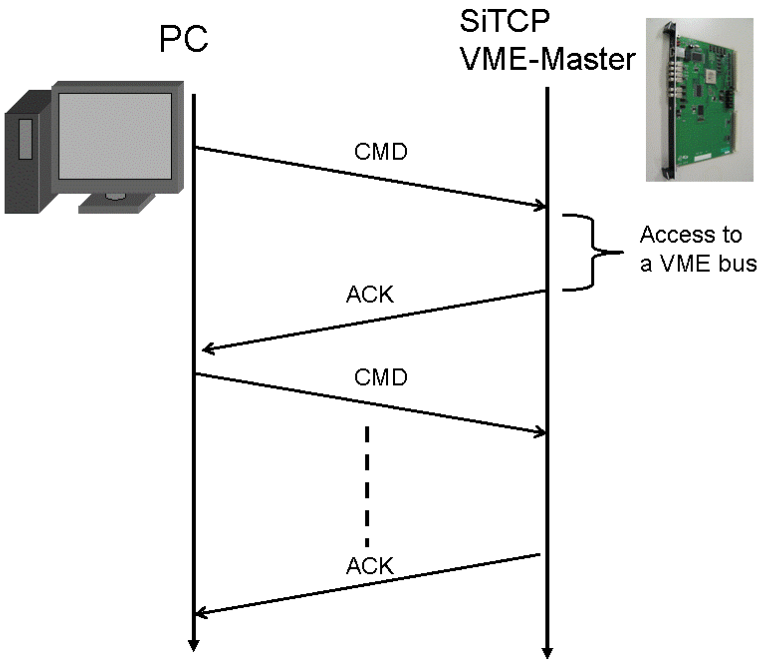


図5 VME アクセス方法

VME アクセスは TCP 上の命令パケットと応答パケットを使用したハンドシェイク方式で行います。図5は PC が SiTCP VME-Master を使用して VME をアクセスした時のパケットを交換の様子を表した図です。時間は図の上から下に流れています。

この図はTCPコネクション確立後のパケットが書かれています。実際に使用するときは、最初にTCPコネクションを確立させることが必要です。通常、TCPコネクションの確立は SOCKET 関数での connect()関数を使用することで行います。

VMEをアクセスする場合は、初めにPC が希望する VME アクセス情報を CMD パケット（詳細は後述）に設定し TCP データとしてモジュールへ通知します。本モジュールは受信したパケットの構文解析を行い指定された VME サイクルを発生させバス・アクセスを行います。アクセスが終了すると応答パケット（ACK パケット）が送り返されてきます。この時、長いデータをリードした場合は適当なサイズに分割され複数パケットが返送されてきます。

ACK パケットには実行したアクセスした内容とアクセス結果が格納されています。アクセスした内容は対応する CMD パケットと同じ内容です。アクセス結果は正常終了か異状終了かがわかるフラグを持っています。フラグの内容は正常終了、パラメータ・エラー、VME バス・タイムアウトなどがあります。パラメータ・エラーは受信した CMD パケットの構文解析中にエラーを検出した時に発生します。エラー発生要因として、対応していないパラメータが設定されている（リザーブ・フィールドに値が設定されているなど）、ヘッダのビットエラー検出に使用している CRC8 の計算結果が合わない、があります。CRC8 はパケットの先頭位置が何らかの理由によりずれた事を検出する為に使用しています。TCP はストリーム型データですから、一度データがずれると先頭を見つけることが出来ません。先頭がずれたままバス・アクセスするとシステムに致命的な問題を引き起こす可能性があります。その問題を回避する為に使用しています。VME バス・タ

イムアウトが発生した場合、どのアドレスをアクセスした時にエラーが発生したかが分かるようにアクセスに成功したバイト数がパケットヘッダに格納されます。また、これらのエラーが発生すると本モジュールはエラー・フラグを持つ ACK パケットを PC に転送した後に TCP コネクションをクローズします（FIN パケットの送信）。TCP FIN を受信したら即座に PC 側も FIN パケットを返送し TCP コネクションを完全に閉じてください。本モジュールがエラー発生により FIN を送付した後に数秒たっても FIN を受信しない場合は RST パケットを PC へ送付して強制的に TCP コネクションを閉じます。これは、誤動作によるシステムへの致命的な問題を回避する為に再度初期状態から再起動するためです。この事により、PC 側では VME アクセス時に問題が起きたことが即座に分かります。問題が起きた原因も返送されてきた ACK パケットを解析することにより知ることが出来ます。

パケット・フォーマット

VME アクセスは TCP 上の命令パケットと応答パケットを使用したハンドシェイク方式で行います。

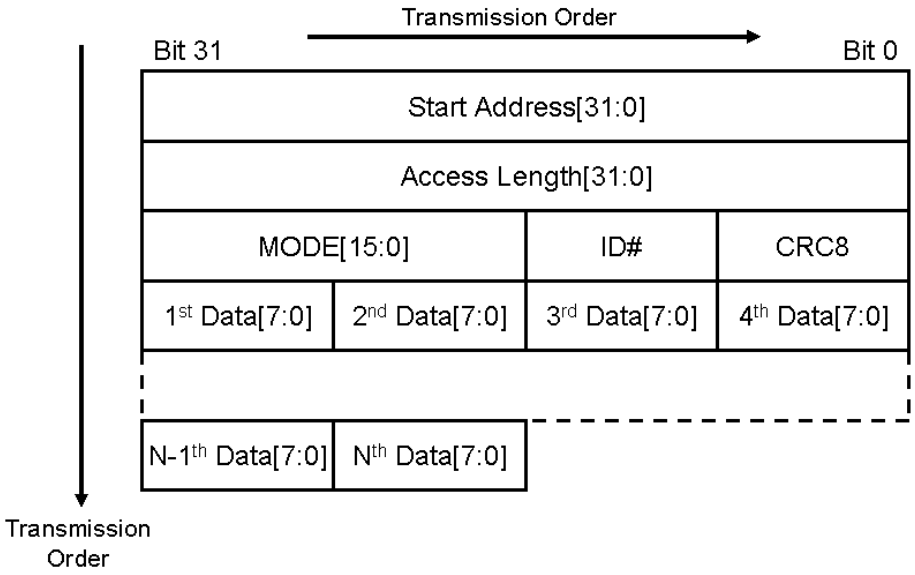


図6　パケット・フォーマット

図6にパケット・フォーマットを、表 1 に MODE フィールドの詳細を示します。

Start Address [31:0]

アクセス先頭アドレスを設定する。アドレスは VME 使用に基づいています（本モジュールでアドレス変換等は行っておりません）。MODE フィールド内のアクセス・モードにより使用するアドレス幅が決まります。例えば A24 を使用すれば、このフィールドの A[23:0]が使用されます。

非整列転送はサポートしていません。Start address はアクセスワード境界、Access length はアクセスワードの倍数を設定する必要があります。例えば、D32 を選択した場合、Start address の下位 2 ビットは必ずゼロ、Access length は4の倍数でなければいけません。

Access Length [31:0]

アクセス長をバイト数で指定します。1 バイトをアクセスする時は 0x1 を設定してください。
MODE フィールド内のアクセス・モード（アクセス・データ幅）により指定できるバイト数が異なりますので注意してください。本モジュールは CMD パケットで設定された内容を VME プロトコルに直接変換する為に制限があります。例えば、データ幅を 16 ビットアクセス指定した場合には奇数バイト数は許されません。奇数バイト数アクセスする場合には 8 ビットアクセスを使用する

か、端数部分以外は 16 ビット・アクセス、端数部分は 8 ビット・アクセスと CMD パケットを分割して使用してください。

非整列転送はサポートしていません。ですから、使用するデータ幅を 1 ワードとして、アクセス長はワードの倍数になります。例えば、D32 を選択した場合、Access length は必ず 4 の倍数でなければいけません。

Mode [15:0]

アクセス・モードを指定します。以下に各ビットの詳細を示します。

表 1　MODE フィールド詳細

Bit	Name	Description
15	Write	Write access=1 Read access=0
14	Echo Write data	Disable echo back of write data = 0, Enable echo back of write data = 1
13	Reserved	Should be 0
12	Reserved	Should be 0
11-10	Data Mode	D8 = 0, D16=1, D32=2 (*1) Value 3 is reserved. Do not use it.
9-8	Address Mode	A16 = 0, A24=1, A32=2 Value 3 is reserved. Do not use it.
7-4	Access Mode	User Data=0, User Prog. =1, Usr BLT =2, Fixed Address and User Data =8, (*2) Fixed Address and User Prog. =9, (*2) Supervisor Data=4, Supervisor Prog. =5, Supervisor BLT =6, Supervisor data & Fixed Address =0xA, (*2) Supervisor prog. & Fixed Address =0xB (*2) Other values are reserved. Do not use those.
3	Acknowledge Flag	Command Packet =0, ACK packet = 1
2	Flag	Active in ACK packets only Normal =0; VME error = 1
1	Flag	Active in ACK packets only Reserved Always 0
0	Flag	Active in ACK packets only Normal = 0, Detected parameter error = 1

(*1) 非整列転送はサポートしていません。アクセス長はアクセスワードの倍数になります。例えば、D32 を選択した場合、Start address の下位 2 ビットは必ずゼロ、Access length は必ず 4 の倍数でなければいけません。
(*2) Fixed Address とは Start Address フィールドに指定したアドレスを Access Length フィールドで指定されたデータ長に相当するだけ、同じアドレスをアクセスします。このモードでBLTおよび非整列転送はサポートされていません。

SiTCP へのアクセス要求は Bit0 から 3 をゼロに、Bit4 から 15 にアクセス方法をセットして行います（コマンド・パケットの発行）。コマンドが実行されると SiTCP から応答パケット（ACK パケット）が返ってきます。

ID

CMD パケットと対応する ACK パケットを認識する為に使用します。
本モジュールは ACK が戻ってくる前に次の CMD パケットを受付待たせる機能があります（コマンド・パイプライン機能）。この機能を使用する場合 CMD パケットと対応する ACK パケットの対応が分からないと制御プログラムが複雑になるため、この複雑さを回避する為に使用します。コマンド・パイプライン機能を使用することにより高速アクセスする事が可能ですが PC の制御プリグラムは複雑になりますので注意してください。

CRC8

パケットヘッダが壊れていないか検査するために使用する CRC コードです。本モジュールはこのコードを使用してパケットの先頭がずれていないかを検査します。エラーを検出した場合はパラメータ・エラー・フラグを持つ ACK パケットを PC へ送ることにより通知します。その後、エラー処理に入ります（VME アクセス概要を参照）。
使用している多項式は x^8+x^2+x+1 です。また、計算する際に CRC 初期値を 0xFF にしています。計算範囲はパケット先頭から 11 バイトです。計算はデータ送信順序で計算します（ビットのスワップ等は行っていません各バイトの bit7 から計算してください）。計算結果をこのフィールドに格納します。格納する際にビット反転させません、計算値をそのまま格納してください。
ACK パケットにも再計算された CRC8 値がこのフィールドに格納されています。再計算する理由は少なくとも MODE フィールドの値が変更される為です。この再計算された CRC8 フィールドをエラー検査に使用することが出来ます。高信頼システムを構築する為に、このフィールドの検査を行ってください。エラーが発生した場合本モジュール内で致命的なエラーが発生した可能性が高いので TCP コネクションを閉じて、再起動してください。もちろん、そのようなエラーが発生する事が無い様に設計されているので、発見時には連絡してください。

Data

ライトコマンドを持つ CMD パケットはライトデータを持つ必要があります。このフィールドはそのデータを格納するフィールドです。格納するバイト数は Access Length フィールドに格納されている値と一致していなければいけません。矛盾がある場合は次のパケット先頭を正しく認識できなくなり、CRC エラーが発生します。

リードコマンドを持つ CMD パケット(MODE[15]=0)にこのフィールドを付けてはいけません。

ライトコマンドの ACK パケットは基本的にはこのフィールドを持ちませんが、MODE 内の Echo Write Data を有効にした場合は正常に書き込まれたデータが格納されます。

リードコマンドに対する ACK パケットの Data フィールドには正常に読み込まれたデータが格納されます。Data フィールドに格納されているデータ長は Access Length フィールドに格納されています。1 が 1 バイトを意味します。例えばエラーが発生し、成功したアクセスがない時には 0 が格納されます。途中で失敗した場合には、正しく読み込むことが出来たデータが格納されます。

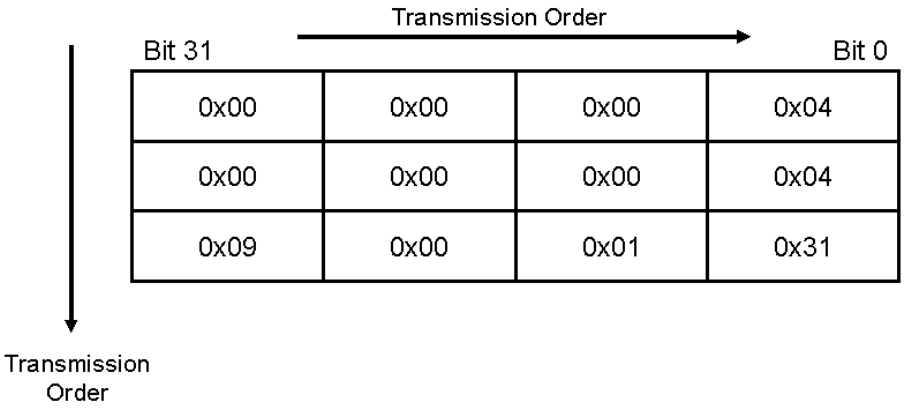
パケット例

以下に幾つかパケットの具体例を挙げます。

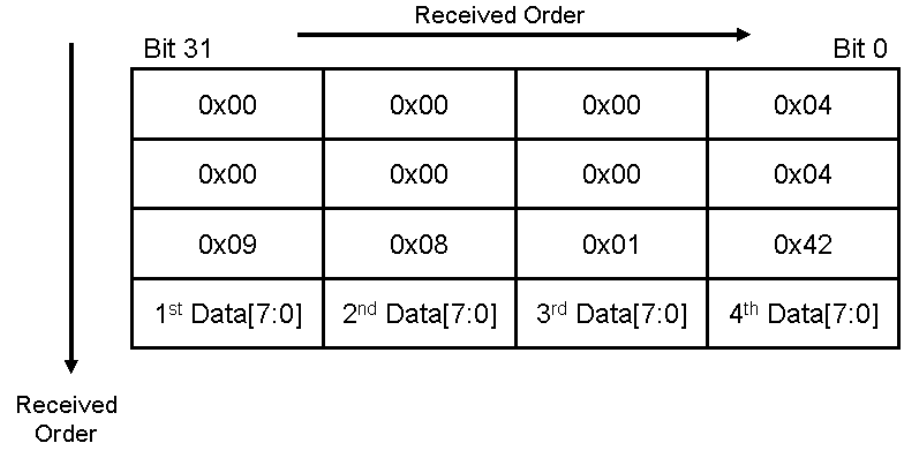
例1

- VME read
- A24
- D32
- User data access

PCから送信する命令パケット



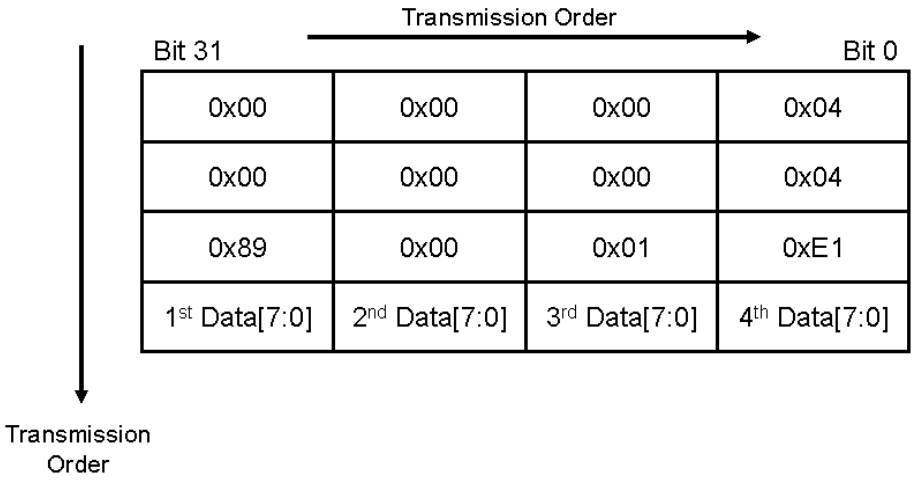
VME-Master module から返送されてくる応答パケット



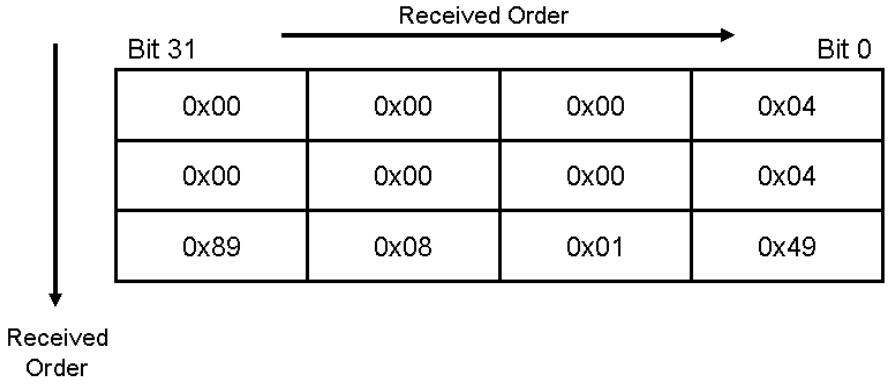
例 2

- VME write
- A24
- D32
- User data access

PCから送信する命令パケット



VME-Master module から返送されてくる応答パケット



参考プログラム

デバックで使用するために作成したプログラムを参考のために以下に示します。

sitcpVmeMaster.h

```
/***********************************************
*                                              *
* SiTCP VME-Master Module                      *
* Header file                                  *
*                                              *
* 2006/03/01 Tomohisa Uchida                   *
*                                              *
***********************************************/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

struct sitcp_vme_master_header{
  unsigned int address;
  unsigned int length;
  unsigned short mode;
  unsigned char id;
  unsigned char crc;
};
```

sitcpVmeMasterClient.c

```
/***********************************************
*                                              *
* Control Program for                          *
* the SiTCP VME-Master Module                  *
*                                              *
* 2008/02/01 Tomohisa Uchida                   *
*                                              *
***********************************************/
#include "sitcpVmeMaster.h"

#define RECV_BUF_SIZE 65536
#define SEND_BUF_SIZE 65536

unsigned char crcCal(unsigned char crc, unsigned char data);

int main(int argc, char* argv[]){

  struct timeval timeout;
  fd_set setSelect;

  /* Check inut values */
  if(argc != 3){
    puts("¥nControl program for a SiTCP VME-Master module.");
    printf("Usage: %s <IP address> <Port #>¥n¥n", argv[0]);
    exit(EXIT_FAILURE);
    }

  char* sitcpIpAddr;
  unsigned int sitcpPort;
```

```
  /* Get IP address and port # of a SiTCP */
  sitcpIpAddr = argv[1];
  sitcpPort   = atoi(argv[2]);


  /* Create a Socket */
  puts("¥nCreate socket...¥n");

  int sock;

  sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

  struct sockaddr_in sitcpAddr;

  sitcpAddr.sin_family      = AF_INET;
  sitcpAddr.sin_port        = htons(sitcpPort);
  sitcpAddr.sin_addr.s_addr = inet_addr(sitcpIpAddr);

  if(connect(sock, (struct sockaddr*) &sitcpAddr, sizeof(sitcpAddr))<0){
    puts("Connect() faild");
    close(sock);
    exit(EXIT_FAILURE);
  }

  puts("Connected...");

  /*********************************************/
  /*        Built a packet                     *
  /*********************************************/
  unsigned char crc;

  unsigned int tempKeyBuf;
  struct sitcp_vme_master_header sndHeader;

  sndHeader.id=0;

  /* Address */
  printf("address (Hex) =");
  scanf("%x",&tempKeyBuf);

  sndHeader.address=htonl(tempKeyBuf);

  unsigned int address;
  address=(unsigned int)tempKeyBuf;

  /* Length */
  printf("Length (Dec) =");
  scanf("%d",&tempKeyBuf);
  sndHeader.length=htonl(tempKeyBuf);

  /* MOde */
  printf("Mode (Hex) =");
  scanf("%x",&tempKeyBuf);

  sndHeader.mode=htons(tempKeyBuf);

  /* ID */
  sndHeader.id++;
  crc=crcCal(crc,(unsigned char)sndHeader.id);

  /* Data */
  unsigned char sndData[SEND_BUF_SIZE-12];
  int tempData;
  int endFlag =0;
  int sndDataLen = 0;

  sndData[0]=0;
```

```
  while(sndDataLen<256 && endFlag==0){
    printf("data[%x] (Hex)=",sndDataLen+address);
    if(scanf("%x",&tempData)!=1) endFlag=1;
    sndData[sndDataLen]=tempData;
    if(tempData<0 || tempData>255) endFlag=1;
    sndDataLen++;
  }

  sndDataLen--;

  unsigned char sndBuf[SEND_BUF_SIZE];

  /* Copy to a tx buffer*/
  memcpy(sndBuf,&sndHeader, sizeof(sndHeader));
  memcpy(sndBuf+sizeof(sndHeader),sndData,sndDataLen);

  /* Calculate CRC8*/
  /* P(x)=x^8 + x^2 + x + 1 */

  int i;

  crc = 0xFF;
  for(i=0;i<11;i++) crc=crcCal(crc,sndBuf[i]);
  printf("CRC (Hex) = %x ¥n", crc);

  sndBuf[i]=crc;

  /* Display Header */
  printf("¥n");
  printf("¥n");
  printf("¥tAddress  : 0x%.8x¥n",ntohl(sndHeader.address));
  printf("¥tLength   : 0x%.8x¥n",ntohl(sndHeader.length));
  printf("¥tMode     : 0x%.4x¥n",ntohs(sndHeader.mode));
  printf("¥tID       : 0x%.2x¥n",sndHeader.id);
  printf("¥tCRC      : 0x%.2x¥n",sndHeader.crc);

  for(i=0; i<sndDataLen; i++){
    printf("¥tData[0x%x]: 0x%.2x¥n",ntohl(sndHeader.address)+i,sndData[i]);
  }
  printf("¥n");

  printf("Send this packet to %s:%d¥n",sitcpIpAddr,sitcpPort);

  char tempKeyStr;

  printf("y(Yes) or n(No) =");
  while(tempKeyStr!='y' && tempKeyStr!='Y' &&
        tempKeyStr!='n' && tempKeyStr!='N'){
    tempKeyStr=getchar();
  }

  if(tempKeyStr=='y' || tempKeyStr=='Y'){
    puts("¥n Tring to sent the packet...¥n");
  }else{
    puts("¥nExit!");
    exit(1);
  }

  int j = 0;

  for(i=0; i< sndDataLen + sizeof(sndHeader);i++){
    if(j==0) {
      printf("¥t[%.3x]:%.2x ",i,sndBuf[i]);
      j++;
    }else if(j==3){
      printf("%.2x¥n",sndBuf[i]);
      j=0;
    }else{
      printf("%.2x ",sndBuf[i]);
```

```
      j++;
    }
  }
  puts("¥n");

  if(send(sock,sndBuf,sndDataLen + sizeof(sndHeader),0)<0){
    puts("send() failed");
    close(sock);
    exit(EXIT_FAILURE);
  }

  puts("---------------------------------------");
  puts("     The packet have been sent!");
  puts("---------------------------------------");

  sndHeader.length=ntohl(sndHeader.length);

  /*************************************************************/
  /*                    Receive packets                        */
  /*************************************************************/

  unsigned char recvBuf[RECV_BUF_SIZE];
  struct sitcp_vme_master_header rcvHeader;

  unsigned int totalRecvLen = 0;
  int detRecvgErr = 0;

  while((sndHeader.length>totalRecvLen) && (detRecvgErr==0)){
    int rBytes = 0;
    int rLen;

    for(rLen = 0; rLen<sizeof(rcvHeader);rLen+=rBytes){
      if((rBytes = recv(sock,&rcvHeader+rLen, sizeof(rcvHeader)-rLen, 0)) <= 0){
        puts(" ");
        puts("Header: recv() faild");
        close(sock);
        exit(EXIT_FAILURE);
      }
    }

    memcpy(recvBuf,&rcvHeader, sizeof(rcvHeader));

    rcvHeader.address = ntohl(rcvHeader.address);
    rcvHeader.length = ntohl(rcvHeader.length);
    rcvHeader.mode = ntohs(rcvHeader.mode);

    totalRecvLen+=rcvHeader.length;

    puts("---------------------------------------");
    puts("    A packet have been received !");
    puts("---------------------------------------");

    printf("   Address = 0x%.8x¥n", rcvHeader.address);
    printf("   Length  = 0x%.8x¥n", rcvHeader.length);
    printf("   Mode    = 0x%.4x¥n", rcvHeader.mode);
    printf("   Id      = 0x%.2x¥n", rcvHeader.id);
    printf("   CRC     = 0x%.2x", rcvHeader.crc);

    /* Caluculate the received CRC */

    crc = 0xFF;
    for(i=0;i<12;i++) crc=crcCal(crc,recvBuf[i]);
    if(crc==0){
      printf(", is CORRECT¥n");
    }else{
      printf(", is NOT CORRECT: %x (Hex)¥n", crc);
      close(sock);
    }
```

```
    /* Error Check */
    if((rcvHeader.mode & 0x1)==0x1){
      puts("¥n  ----- Parameter Error !! -----");
      detRecvgErr=1;
    }
    if((rcvHeader.mode & 0x4)==0x4){
      puts("¥n  ----- VME bus TIMEOUT !! -----");
      detRecvgErr=1;
    }

    /* Read read data from the VME*/
    if(((rcvHeader.mode & 0xC00F)==0xC008) ||
       ((rcvHeader.mode & 0xC00F)==0x0008) ||
       ((rcvHeader.mode & 0xC00F)==0x000C)){

      rBytes=0;

      for(rLen = 0; rLen<rcvHeader.length;rLen+=rBytes){
         if((rBytes = recv(sock,recvBuf+rLen, rcvHeader.length-rLen, 0)) <= 0){
           puts(" ");
           puts("Read Data: recv() faild");
           close(sock);
           exit(EXIT_FAILURE);
         }
      }

      j=0;

      puts("");

      for(i=0;i<rcvHeader.length;i++){
         if(j==0) {
           printf("   [%.8x]  %.2x ",rcvHeader.address+i,recvBuf[i]);
           j++;
         }else if(j==3){
           printf("%.2x¥n",recvBuf[i]);
           j=0;
         }else{
           printf("%.2x ",recvBuf[i]);
           j++;
         }
      }
      puts("");
    }

    puts("---------------------------------------");
  }

  sleep(1);

  printf("¥n***** Success %d/%d *****¥n",totalRecvLen,sndHeader.length);

  close(sock);
}

unsigned char crcCal(unsigned char crc, unsigned char data){
  unsigned char crcReg[9];
  unsigned char inBit;

  int i, j;
  unsigned char crcMask = 1;

  for(i=0; i<8; i++){
    crcReg[i]=crc;
    crcReg[i]&=crcMask;
    if(crcReg[i]!=0) crcReg[i]=0xFF;
    crcMask<<=1;
  }
```

```
  for(i=0; i<8; i++){
    inBit=data & 0x80;

    if(inBit!=0)inBit=0xFF;

    crcReg[8]=inBit^crcReg[7];

    for(j=7; j>0; j--){
      if(j<3){
        crcReg[j]=crcReg[j-1]^crcReg[8];
      }else{
        crcReg[j]=crcReg[j-1];
      }
    }
    crcReg[0]=crcReg[8];
    data<<=1;
  }

  crc=0;
  crcMask=1;

  for(i=0; i<8; i++){
    if(crcReg[i]!=0) crc|=crcMask;
    crcMask<<=1;
  }
  return(crc);
}
```

本マニュアルおよび製品に関するお問い合わせ

株式会社　Bee Beans Technologies
茨城県つくば市千現 2-1-6
つくば研究支援センター1 階 B-5
TEL:029-858-2484　FAX:029-858-2543
e-mail : tech-support@bbtech.co.jp
URL：http://bbtech.co.jp/