ACS Tech80

REDEFINING MOTION CONTROL

Model 5312B

4 Axis Quadrature Encoder - PC

Software Guide

ACS-Tech80 Part Number 700020

Document version no. 1.30

Intentionally Left Blank

Document version no 1.30 (May 2001)

Part number: 700020

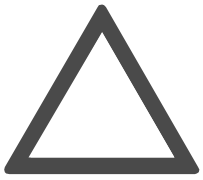
ACS Tech80

Internet: http://www.acs-tech80.com/
E-mail: info@acs-tech80.com
support@acs-tech80.com

ACS-Tech80 Inc.
7351 Kirkwood Lane North
Suite 130
Maple Grove, MN, 55427, USA
Tel: +1.763.493.4080
Fax: +1.763.493.4089

ACS-Tech80 Ltd.
Ramat Gabriel Industrial Park,
POB 5668
Migdal Ha'Emek, 10500, Israel
Tel: +972.6.6546440
Fax: +972.6.6546443

Warning



Dangerous voltages are present in this equipment!
Contact with live parts could cause serious injury or death!
Refer connection, installation, maintenance, adjustment, servicing and operation to qualified personnel.

Intentionally Left Blank

Table of Contents

About this Manual

Quality Control

ACS-Tech80 manufactures quality and versatile products, and we want our documentation to reflect that same quality. We take great pains to publish manuals that are informative and well organized. We also strive to make our documentation easy to understand for the novice as well as the expert.

If you have comments or suggestions about how to make this (or other) manuals easier to understand, or if you find an error or an omission, please email us at support@acs-tech80.com, You will receive a complimentary updated manual.

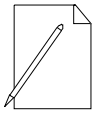
Procedural Conventions

ACS-Tech80 uses various conventions throughout this and all other manuals. You should become familiar with these conventions as they are used to draw attention to items of importance and items that will generally assist you in understanding a particular area.

WARNING

This type of box is used to indicate that an action must be done with great care. Otherwise personal injury or significant equipment damage may result.

NOTE



This type of box is used to indicate important information needed to perform an action or information that is *nice-to-know*.

CAUTION

This type of box is used to indicate that an action may cause minor equipment damage or the loss of data if not performed carefully.

When referring to pin numbering, pin 1 is *always* associated with a square solder pad on the actual component footprint.

Notational Conventions

A forward slash *(I)* preceding a signal name denotes an active LOW signal. This is a standard Intel convention.

Caret brackets (<>) denote keystrokes. For instance <Enter> represents carriage—return—with—line—feed keystroke, and <Esc> represents an escape keystroke.

Driver routine declarations are shown for C and BASIC (where applicable).

Hungarian notation is used for software parameters. In other words, the parameter type is denoted by a one or two letter small case prefix:

c	character, signed or unsigned
s	short integer, signed
w	short integer, unsigned
l	long integer, signed
dw	long integer, unsigned

For example, *wBoardAddr* would be an unsigned short integer parameter.

An additional p prefix before the type prefix indicates that the parameter is being passed by reference instead of by value. (A pointer to the variable is being passed instead of the variable itself.)

For example, *pwErr* would be an unsigned short integer parameter passed by reference.

This notation is also used in BASIC although no distinction between signed and unsigned variables exists.

In BASIC, all parameters also have a type suffix:

$	character, signed or unsigned
%	integer, signed or unsigned
&	long integer, signed or unsigned

Routine names are printed in bold plain font when they appear outside of function declarations, e.g., **te53l2InitSw**.

Parameter names are printed in italics when they appear outside of function declarations, e.g., *sControls.*

Constants are defined with all caps, e.g., ALL_AXES. Underscores (_) must be replaced by periods (.) for use with BASIC.

Combinational logic and hexadecimal notation is in C convention in many cases. For example, the hexadecimal number 7Ch is shown as 0x7C.

C relational operators for OR and AND functions— "| |" and "&&"— are used to minimize the confusion associated with grammar.

How This Book Is Organized

Front Section

A TOC (Table of Contents) lists all the sections, appendices, and attachments in the manual set. In addition, the TOC lists all figures, tables, and sample code where applicable.

Section 1

Programming Overview

This section offers a general discussion on how to compile and link the M5312 software drivers with an assortment of compiler resources.

Section 2

Example Programs

This section lists sample programs for typical applications in C, BASIC, and Pascal.

Section 3

Interrupt Handling

This section shows how to set up your program to use the software driver to generate interrupts.

Section 5

Alphabetical Routine Summary

This section lists all the software driver routines in alphabetical order.

Section 6

Driver Routine Descriptions

This appendix is a complete reference for the M5312 software driver routines.

Section 7

Demonstration Program

This appendix familiarizes you with a simple demo program that configures the board and reads back encoder counts on all 4 axes independently.

1 Programming Overview

1.1 Installing the Model 5312 Software

The Model 5312 driver includes the batch file, INSTALL.BAT, to install the software. The batch file takes one argument—the path where you will install the software. For example, to install the software on the C drive into a subdirectory called 5312, type on the command line:

```
install c:\5312
```

We recommend you use the same path for the installation of all drivers. This puts all include files, examples, etc., together. This is especially important when using QuickBASIC, where you will have to combine many libraries into a quick library.

BASIC, C, and Pascal subdirectories will be created in the directory you specify. You may delete any unneeded subdirectories to save disk space.

1.2 Compiling and Linking

This section shows how to compile a program using the Model 5312 driver with the various supported compilers. We assume the source file is named DEMO.C for C, DEMO.BAS for BASIC, and DEMO.PAS for Pascal. The following notational convention is useful.

x is a variable place holder for library size denoting:

s	*small* model,
m	*medium* model,
c	*compact* model,
l	*large* model
h	*huge* model

1.2.1 Microsoft C or Microsoft QuickC

To compile and link on the command line:

```
cl /Ax /Gs demo.c  te5312x.lib        (C)
qcl /Ax /Gs demo.c  te5312x.lib       (QuickC)
```

For CodeView compatibility, include the `/Zi` switch.

To use the Model 5312 driver in the QuickC environment, take the following steps:

1.) In the *Make* menu, select the *Set Program List* option.

2.) After naming the Make file, select *Edit Program List*, and enter the names of the source file (DEMO.C) and the appropriate library (e.g. te5312s.lib for *small* model).

3.) In the *Options ! Make* menu, select the *Compiler Flags* option and set the appropriate memory model (this model must match the library in the make list). If you use interrupts, turn stack-checking off.

1.2.2 Borland or Turbo C/C++

To compile and link on the command line:

```
tcc -mx demo.c te5312x.lib      (Turbo C)
bcc -mx demo.o te5312x.lib      (Borland C)
```

For Turbo Debugger compatibility, include the `-v` option. To use the Model 5312 driver in the Borland environment, take the following steps:

1.) In the *Project/Open Project* menu, type in the name of the project file you want to create.

2.) In the *Project/Add Item* menu, enter the names of the source file (DEMO.C) and the appropriate library (e.g. te5312s.lib for small model).

3.) In the *Options/Compiler/Code Generation* menu, set the appropriate memory model (this model must match the library in the *Make* list).

1.2.3 Microsoft QuickBASIC

If you use compiled BASIC, exclusively, and never program in the QuickBASIC environment, you can link the library te5312b.lib into your application.

```
bc demo.bas;
link demo.obj,,,te5312b. lib
```

To compile and link for CodeView compatibility, type:

```
bc /Zi demo.bas;
link /CO demo.obj,,,te5312b.lib
```

If you use the QuickBASIC environment, you first have to run the batch file LB5312.BAT. This batch file will need modification, depending on which QuickBASIC version you use. The necessary modifications are explained by the remarks in the batch file itself.

The batch file creates two files—te53l2qb.qlb and te53l2qb.lib. te5312qb.qlb is a quick library for use in the QuickBASIC environment and te5312qb.lib is the command line equivalent.

In other words, develop your program with teS3l2qb.qlb and then in the final compilation, link with te53l2qb.lib.

Then, to use the Model 5312 driver in the QuickBASIC environment, type:

```
qb demo.bas /lte5312qb.qlb
```

To compile on the command line:

```
bc demo.bas;
link demo.obj,,,te5312qb.lib
```

To compile and link for CodeView compatibility:

```
bc /Zi demo.bas;
link /CO demo.obj,,,te5312qb.lib
```

The libraries—te5312b.lib and te53l2qb.lib—are similar but not identical. Library, te5312b.lib, calls two routines not contained in the library itself: **MoveDone** and **InputAlert**. These two routines have to be included in your source code if you need to link te5312b.lib into application program. The file, INTR5312.BAS, contains stub versions of these routines that you can use as a guide, or you can compile and link the file itself into the application. Since te5312b.lib has unresolved references, it can not be converted into a quick library.

On the other hand, te5312qb.lib, is created by the batch file by compiling INTR5312.BAS and linking the resulting object file with te5312b.lib. It has no unresolved references and can be converted into the quick library te53l2qb.qlb. A program developed in the QuickBASIC environment, using te53l2qb.qlb can be compiled on the command line and linked with te53l2qb.lib without modifying the source code. See the section on using interrupts with BASIC for more information.

1.2.4 Borland Turbo Pascal

To compile and link on the command line, type:

```
tpc /$S- demo
```

To compile for Turbo Debugger compatibility, include the `/v` option.

To use the Model 5312 driver in the Turbo Pascal environment, type:

```
turbo demo
```

The source file must include the line: `uses te5312p;`.

1.3 Programming in DOS

Reminder: DOS is not reentrant. Hence, if you use interrupts, be sure to turn stack checking off. To turn off stack checking in Microsoft Cor Quick C, include the `/Gs` switch (option); in Turbo Pascal include `/$S` on the command line or include the line `{$S–}` in the program source code. See the section on using interrupts with BASIC for more information.

1.4 Programming in Windows

To program in Windows, you need to link the Model 5312 standard libraries to 16-bit dynamic linking libraries (DLLs). These are files with the extension .dll located in the appropriate subdirectories in your specified main directory upon installation. (The corresponding files in the DOS operating system are .lib files.) They enable you to operate in Windows 3.1 programming environments.

1.4.1 Procedure

Windows enables you to easily link to these DLLs by following these steps.

1. Upon installation, move all DLL files installed in your specified directory into the Windows \ System directory on your hard drive. Though these DLLs may work from your working directory, they work better from the Windows \ System directory.

2. Include the line `#include te5312w.dll` in the header file of your program.

3. When your program calls any one of the functions in the DLL, Windows will automatically link to the DLL in your Windows \ System directory.

Because of the dynamic nature of DLLs, in many ways, linking is easier in Windows than in DOS. Thus, no further action is necessary.

Note: te5312w.dll contains all functions contained in te5312x.lib *except* for interrupt-handling routines. The reason is that interrupts are not recommended in Windows for two reasons, both related to interrupt latencies. First, on average, interrupt latencies in Windows are seven times that of DOS. Second, latencies in Windows are much more varied than in DOS. That means that you may experience latencies in Windows between four and ten times that of DOS latencies. The unpredictable nature of interrupt latencies in Windows can be fatal to the timing of your system. Hence we do not include interrupt-handling functions in our DLLs.

2 Example Programs

2.1 Program In C

```
#include "te5312.h"
#include <stdio.h>
#include <conio.h>

#define BOARD  0
#define AXIS_A 0
#define AXIS_B 1
#define GLOBAL -1

// interrupt hook prototypes
static void te5312IndexAlert(short *psAxisNum);
static void te5312WrapAroundAlert(short *psAxisNum);

// interrupt counters
static unsigned short wCarryA, wCarryB;
static unsigned short wIndexA, wlndexB;

void main()
{
  unsigned short wBoardAddr;
  long lCntA, lCntB;
  short sStatA, sStatB;
  short sIRQNum;

  // get the address
  printf("\nEnter the base address the 5312 is strapped "
    "at in hexadecimal -  ");
  scanf("%x", &wBoardAddr);

  // get the IRQ number
  do{
    printf("\nEnter the interrupt request line used (2 to 7) - ");
    scanf(''%u", &sIRQNum);
  }while((sIRQNum < 2) || (sIRQNum > 7));

  // initialize the software
  te53l2InitSw();

  // initialize the board (assume the board has at least two axes)
  te5312InitBoard(wBoardAddr, 2);

  // zero the counters
  te5312LoadCntr(GLOBAL, 0L);

  // initialize interrupts
  te5312InterruptHooks(te53l2WrapAroundAlert, te53l2lndexAlert);
  te5312EnableIRQ(BOARD, sIRQNum);
  te5312lndexAlertOn(GLOBAL);
  te5312WrapAroundAlertOn(GLOBAL);
```

```
  //print column headers
  printf("\nPress any key to exit\n\n"
    "             Axis A                  "
    "             Axis B\n                "
    "                   Index WrapAround  "
    "                   Index WrapAround\n"
    " Count Status Interrupts Interrupts  "
    " Count Status Interrupts Interrupts\n");

  //display counter values and status until key pressed
  while(!kbhit()){
    lCntA = te53l2ReadCntr(AXIS_A); ICntB = te5312ReadCntr (Axis_B);
    sStatA = te53l2HeadSts(AXIS_A); sStatB = te5312ReadSts (Axis_B);
    printf'("\r%8ld  %2X    %5u    %5u    ", lCntA, StatA,
      wIndexA, wCarryA);
    printf("%8Id  %2X    %5u    %5u", ICntB, sStatB,
      wIndexB, wCarryB);
  }
  if (!getch())
    (void)getch()
  printf("\n");
  // disable interrupts before exiting program
  te5312DisableIRQ();
}
void te5312WrapAroundAlert(short *psAxisNum)
{
  switch(*psAxisNum){
  case AXIS_A: wcarryA++; break;
  case AXIS_B: wcarryB++; break;
  }
}

void te5312IndexAlert(short *psAxisNum)
{
  switch (*psAxisNum){
  caseAXIS_A: wIndexA++; break;
  caseAXIS_B: wIndexB++; break;
  }
}
```

2.2 Program in BASIC

```
'$INCLUDE: 'TE5312.BAS'

CONST BOARD = 0
CONST GLOBAL = -1
CONST AXIS.A = 0
CONST AXIS.B = 1
CONST BOARD.ADDR = &H020A
CONST NUM.AXES = 2
CONST IRQ.NUM = 2

'Declare Global Variables
COMMON SHARED CarryA%, CarryB%
COMMON SHARED IndexA%, IndexB%

REM initialize the software
version% = te5312InitSw
cls
print "VERSION NUMBER = "; HEX$(version%)

REM initialize the board
x% = te5312InitBoard(BOARD.ADDR, NUM.AXES)

REM zero the counters
x% = te5312LoadCntr(-1, 0)

REM initialize interrupts
x% = te5312EnableIRQ(BOARD, IRQ.NUM)
x% = te5312lndexAlertOn(-1)
x% = te5312WrapAroundAlertOn(-1)
print
print "Press any key to exit"
print

REM display counter values and status until key pressed
do
  locate 5, 1
  CntA& = te5312ReadCntr(AXIS.A)
  CntB& = te5312ReadCntr(AXIS.B)
  StatA% = te5312ReadSts(AXIS.A)
  StatB% = te5312ReadSts(AXIS.B)
  print "Axis A"
  print " Count = "; CntA&; "          "
  print " Status = "; HEX$(StatA%) "    "
  print " Index Interrupts = "; IndexA%
  print " Wrap-Around Interrupts = "; CarryA%
  print
```

```
  print "Axis B"
  print " Count = "; CntB&; "          "
  print " Status = "; HEX$(StatB%); "    "
  print " Index Interrupts = "; IndexB%
  print " Wrap-Around Interrupts = ": CarryB%
  A$ = INKEY$

loop while LEN(A$) = 0

REM disable interrupts before exciting program
X% =  te5312DisableIRQ

REM If using this file in QuickBasic, move the rest of this file
REM to the file INTR5312.BAS and remove the remark notations from
REM the beginning of the following two declaration lines. Then
REM run the batch file QLB5312.BAT:
REM '$INCLUDE: 'TE5312.BAS'
REM DIM SHARED IndexA%, IndexB%, CarryA%; CarryB%

SUB te5312IndexAlert (AxisNum%)
  if (AxisNum% = AXIS.A) then
    IndexA% = IndexA% + 1
  elseif (AxisNum% = AXIS.B) then
    IndexB% = IndexB% + 1
  endif
END SUB

SUB  te5312WrapAroundAlert (AxisNum%)
  if (AxisNum% = AXIS.A) then
    CarryA% = CarryA% + 1
  elseif (AxisNum% = AXIS.B) then
    CarryB% = CarryB% + 1
  endif
ENDSUB
```

2.3 Program in Pascal

```
Program example1;

uses te5312p, crt;

const
  { Define some initial constants }
  ADDR = $20A;     { board address }
  NUM_AXES = 2;    { number of axes on board }
  IRQ = 3;         { IRQ number }
  BOARD =0;        { board number }
  GLOBAL = -1;     { Global number }
  AXIS_A = 0;      { first axis number to be moved }
  AXIS_B = 1;      { second axis number to be moved }
  CR = #13;        { carriage return }

var
  { interrupt counters }
  wCarryA, wCarryB, wIndexA, wIndexB : word;
  wBoardAddr : word;
  lCntA, lCntB : longint;
  sStatA, sStatB : integer;
  sTemp : integer;

{$S-}          { turn stack checking off for interrupts }

procedure te5312WrapAroundAlert (var psAxisNum : integer ); far;
begin
  if (psAxisNum = AXIS_A) then wCarryA  := wCarryA + 1;
  if (psAxisNum = AXIS_B) then wCarryB  := wCarryB + 1;
end;

procedure  te5312IndexAlert (var psAxisNum : integer); far;
begin
  if (psAxisNum = AXIS_A) then wIndexA  := wIndexA + 1;
  if (psAxisNum = AXIS_B) then wIndexB  := wIndexB + 1;
end;

begin
  { initialize the software }
  sTemp  := te5312InitSw;

  { initialize the board }
  sTemp := te5312InitBoard(ADDR, NUM_AXES);

  { zero the counters }
  sTemp := te5312LoadCntr(GLOBAL, 0);
```

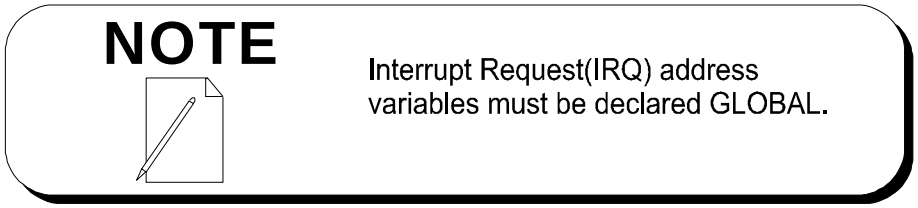
```
  { initialize interrupts }
  wIndexA := 0; wIndexB := 0;
  wCarryA := 0; wCarryB := 0;
  InterruptHooks(te5312WrapAroundAlert, te5312IndexAlert);
  sTemp := te5312EnableIRQ(BOARD, IRQ);
  sTemp := te53l2IndexAlertOn(GLOBAL);
  sTemp := te5312WrapAroundAlertOn(GLOBAL);

  { print column headers }
  Writeln('Press any key to exit');
  Writeln;
  Write('            Axis A                    ');
  Writeln('            Axis B');
  Write('               Index     WrapAround   ');
  Writeln('               Index      WrapAround');
  Write('   Count  Status Interrupts Interrupts ');
  Writeln('   Count  Status Interrupts Interrupts');
  { display counter values and status until key pressed }
  while(not KeyPressed) do
  begin
    lCntA := te5312BeadCntr(AXIS_A);
    lCntB := te5312ReadCntr(AXIS_B);
    eStatA := te5312ReadSts(AXIS_A);
    eStatB := te5312ReadSts(AXIS_B);
    Write(CR);
    Write(lCntA : 8, sStatA : 5, wIndexA :10, wCarryA : 11);
    Write(lCntB : 13, sStatB : 5, wIndexB :10, wCarryB : 11);
  end;
  Writeln;
  { disable interrupts before exiting program }
  sTemp := te5312DisableIRQ;
end.
```

3 Interrupt Handling

3.1 Introduction

The 5312 driver greatly simplifies the use of interrupts. When an interrupt occurs, the driver handles all interrupt overhead and then calls your routines to act on the interrupts.



3.1.1 Enabling Interrupts

The first routine you need to call is **te5312EnableIRQ** before interrupts can be used. At the end of the program, call **te5312DisableIRQ** to restore the interrupt vectors and interrupt masks to their original state. You need to supply two routines to handle the two interrupt sources—overflow/underflow and index valid. The two routines are described below.

For BASIC, the names given below are fixed. The linker will expect to find two routines with these names. For C or Pascal the routines can be named anything because the *address* rather than the *name* of each routine is passed to the **te5312InterruptHooks** routine.

te5312WrapAroundAlert — This routine will be called when the encoder generates either a borrow or a carry. It will receive one argument by reference—the axis number of the encoder causing the interrupt.

te5312IndexAlert — This routine will be called when the index input goes active. It will receive one argument by reference—the axis number corresponding to the index input causing the interrupt.

3.2 Interrupts in C or Pascal

The example programs in section 2 show how interrupts are set up. Interrupt hook routines are installed by calling **te5312InterruptHooks**. A warning will be generated if you attempt to install improper routines (routines that do not accept the proper number and type of arguments). Remember to turn off stack checking for the interrupt hook functions and any routines they call.

3.3 Interrupts in BASIC

The example program given earlier shows how interrupts are used. You must provide two routines—**te5312WrapAroundAlert** and **te5312IndexAlert**.

One note about using interrupts in the QuickBASIC environment: it can be done, but the interrupt handling routines must be in the Quick Library te5312qb.qlb. To do this, use the file INTR5312.BAS to write your interrupt hook routines. Then run the batch file QLB5312.BAT to compile INTR5312.BAS and add it to the libraries—te53l2qb.qlb and te53l2qb.lib. The library te53l2qb.lib is an alternative to using te5312b.lib and is supplied to provide a command line equivalent library to the Quick Library. You can develop a program in the environment with the Quick Library and then compile and link on the command line without modification. If you use te5312b.lib, you will have to add your interrupt hook routines to the source file before compiling.

3.4 General Notes on Using Interrupts

There are some cautions to be aware of when using interrupts:

DOS is not re-entrant. If an interrupt is generated while in a DOS call, the interrupt routine can not call another DOS function. With Basic, C, and Pascal, DOS is usually used for screen output, keyboard input, and disk and file I/O. We recommend you not use DOS in your interrupt routines. One method for avoiding this is to set a global flag in your interrupt routine, and then have the main routine check this flag and call DOS when the flag is set.

For example, if you wanted to print a message when an interrupt occurred, the interrupt routine sets a flag. Then when the main program sees the flag set, it will print the message.

Turn off stack checking when using interrupts with C. If you encounter a stack overflow, stack checking is not turned off. Check the compiler manual for instructions on how to do this.

4 Alphabetical Routine Summary

The 5312 driver software consists of the following routines. A more complete description of each is given in Section 5, Driver Routine Descriptions.

Prefix	Variable Type
c	character, signed or unsigned
s	short integer, signed
w	short integer, unsigned
l	long integer, signed
dw	long integer, unsigned
p	pointer

te5312DisableIRQ( )	Restore old interrupt vectors and disables PC IRQ lines.
te5312EnableIRQ(*wBoardNum, sIRQLevel*)	Set up interrupt vector and enable PC IRQ line on the bus.
te5312IndexAlertOff(*sAxisNum*)	Disable index interrupt.
te5312IndexAlertOn(*wAxisNum*)	Enable index interrupt.
te5312InitBoard(*wBoardAddr, wNumAxes*)	Initialize 5312 board.
te5312InitEncoder(*sAxisNum, sMCR sICR, sOCCR, sQR*)	Initialize encoder.
te5312InitSw( )	Initialize software.
te5312InterruptHooks(**WrapAroundHook, *IndexHook*)	Define hooks to user functions called on interrupts (not usable in BASIC).
te5312LoadCntr(*sAxisNum, lValue*)	Load encoder counter.
te5312LoadPr(*sAxisNum, lValue*)	Load encoder preset register.
te5312ReadCntr(*sAxisNum*)	Read encoder counter.
te5312ReadOL(*sAxisNum*)	Read encoder output latch.
te5312ReadSts(*sAxisNum*)	Read encoder status.

te5312WrapAroundAlertOff(*sAxisNum*)	Disable borrow/carry interrupt.
te5312WrapAroundAlertOn(*sAxisNum*)	Enable borrow/carry interrupt.
te5312WriteCmd(*sAxisNum, sCmd*)	Write encoder command.

5 Driver Routine Descriptions

Notational Conventions

The declarations for each routine is shown for C , BASIC, and Pascal.

In C or Pascal, the type of a parameter is denoted by its one letter small-case prefix:

Prefix	Variable Type
c	Character, signed or unsigned
s	Short integer, signed
w	Short integer, unsigned
I	Long integer, signed
dw	Long integer, unsigned
p	Pointer

F or instance, s*AxisNum* indicates that this variable is a unsigned short integer.

In BASIC, the type of a parameter is always explicitly indicated by a type suffix:

Prefix	variable Type
%	Short integer, signed or unsigned
&	Long integer, signed or unsigned
$	Character, signed or unsigned

For instance, *AxisNum%* indicates the this variable is a short integer.

Routine names are printed in bold sans serif font, **te53l2InitSw**.

Parameter names are printed in italics, *sAxisNum*.

Constants are defined with all caps, TE5312CMD_QR. Underscores must be replaced by periods for use with BASIC.

te5312DisableIRQ

Disable Interrupt Request

Declarations:

C: `short te53l2DisableIRQ(void);`

BASIC: `DECLARE FUNCTION te5312DisableIRQ%()`

Pascal: `function te53l2DisableIRQ : integer;`

Description:

This routine masks the IRQ lines chosen with **te5312EnableIRQ** calls and restores the corresponding interrupt vectors to their original values. If you call **te5312EnableIRQ** at least once, call **te5312DisableIRQ** before exiting the program.

Return Code:

(0) No error.

See Also:

te5312EnableIRQ

te5312EnableIRQ

Enable Interrupt Request

Declarations:

C:
```
short te5312EnableIRQ(unsigned short wBoardNum, short
sIRQLevel);
```

BASIC:
```
DECLARE FUNCTION te5312EnableIRQ%(BYVAL BoardNum%, BYVAL
IRQLevel%)
```

Pascal:
```
function te5312EnableIRQ(wBoardNum : word; sIRQLevel :
integer) : integer;
```

Description:

This routine reassigns the appropriate interrupt vector to point to the driver interrupt handler for the specified board, saving the old vector. It also unmasks the interrupt on the PC.

Each board must use a different IRQ number. The old vectors can later be restored with the **te5312DisableIRQ** routine.

Parameters:

BoardNum Board number (0 to 5).

IRQLevel IRQ number (2 to 7).

Return Code:

(0) No error.

(-1) Invalid board number or IRQ number or the IRQ number has previously been assigned to another board.

See Also:

te5312EnableIRQ

te5312IndexAlertOff

Disable Index Interrupt

Declarations:

C: `short te5312IndexAlertOff(short sAxisNum);`

BASIC: `DECLARE FUNCTION te5312IndexAlertOff%(BYVAL AxisNum%)`

Pascal: `function te5312IndexAlertOff(sAxisNum : integer) : integer;`

Description:

This routine disables the index input for the specified axis from causing an interrupt when index goes active. The index input can be either active HIGH or active LOW depending on jumper settings—W13, W16, W50, and W51.

Parameters:

AxisNum Axis number (0 to 23) or global number (-1 to -6).

Return Code:

(0) No error.
(-1) Invalid axis number.

See Also:

te5312IndexAlertOn

te5312IndexAlertOn

Enable Index Interrupt

Declarations:

C: `short te5312IndexAlertOn(short sAxisNum);`

BASIC: `DECLARE FUNCTION te5312IndexAlertOn%(BYVAL AxisNum%)`

Pascal: `function te5312IndexAlertOn(sAxisNum : integer ) : integer;`

Description:

This routine enables the index input for the specified axis to cause an interrupt when the input goes active. The index input can be either active HIGH or active LOW depending on jumper settings—W13, W16, W50, and W51.

Parameters:

AxisNum Axis number (0 to 23) or global number (-1 to -6).

Return Code:

(0) No error.

(-1) Invalid axis number.

See Also:

te5312IndexAlertOff

te5312InitBoard

Initialize Board

Declarations:

C:
```
short te5312InitBoard(unsigned short wBoardAddr, unsigned
short wNumAxes);
```

BASIC:
```
DECLARE FUNCTION  te5312InitBoard%(BYVAL BoardAddr%, BYVAL
NumAxes%)
```

Pascal:
```
function te5312InitBoard(wBoardAddr, wNumAxes : word) :
integer;
```

Description:

This routine initializes a 5312 board jumpered to the given address. Call the routine **te53l2lnitSw** first to initialize the software, then call **te53l2lnitBoard** once for every 5312 board in the system.

Each board in the system will be sequentially assigned a board number from 0 to 5 used to identify the board in calls to other routines. Likewise, each encoder in the system will be sequentially assigned an *axis number* from 0 to 23. Each board will be assigned from 0 to 4 axis numbers depending on how many encoders are specified on the board.

Each board will also be assigned *a global number*, from —1 to —6. You can use a global number in place of an axis number in routines that write to an encoder. In this case, all encoders on a corresponding board will be serviced at the same time.

te53l2lnitBoard initializes the board interrupt controller and each encoder. For each encoder, **te53l2lnitBoard** resets the Master Control Register (MCR) to:

TE5312MCR_ADDR_RST | |

TE5312MCR_FLAG_RST | |

TE5312MCR_CMP_RST | |

TE5312MCR_MASTER_RESET.

The Input Control Register (ICR) is set to the constant TE5312ICR_ENABLE enabling the phase inputs.

The Output / Counter Control Register (OCCR) is cleared to zero.

The Quadrature Register (QR) is set to the constant TE5312QR_X4 putting the board into 4*x* quadrature mode.

You can override this call by calling **te53l2InitEncoder**.

Parameters:

BoardAddr Address of the 5312 board.
NumAxes Number of encoders on the board.

Return Code:

(0) No error.

(-1) Too many boards initialized, invalid board address, or invalid number of axes.

(>0) One or more axes failed initialization. If bit zero is set, the first axis failed initialization; if bit one is set, the second axis failed initialization, etc.

An axis is assigned an axis number even if it fails initialization.

te5312InitEncoder

Initialize Encoder

Declarations:

C:
```
short  te5312InitEncoder(short sAxisNum, short sMCR, short
sICR, short sOCCR, short sQR);
```

BASIC:
```
DECLARE FUNCTION te5312InitEncoder%(BYVAL AxisNum%, BYVAL
MCR%, BYVAL ICR%, BYVAL OCCR%, BYVAL QR%)
```

Pascal:
```
function  te5312InitEncoder(sAxisNum; sMCR, sICH, sOCCR,
sQR : integer) : integer;
```

Description:

This routine writes the specified commands to the four command registers of an axis.

Parameters:

AxisNum Axis number (0 to 23) or global number (-1 to -6).

MCR Master Control Register command to be written.

ICR Input Control Register command to be written.

OCCR Output /Counter Control Register command to be written.

QR Quadrature Control command to be written.

For *MCR,* OR any of the following constants together:

TE5312MCR_ADDR_RST	Resets the address counters.
TE5312MCR_CNT_OL	Loads the Output Latch with the counter value.
TE5312MCR_FLAG_RST	Zeroes the counter, resets the borrow & carry flags, and sets the sign flag.

TE5312MCR_PR_CNT	Loads the counter with the value of the Preset Register.
TE5312MCR_CMP_RST	Resets the compare flag.
TE5312MCR_MASTER_RST	Does a master reset.

For *ICR,* OR any of the following constants together:

TE5312ICR_DIR	Phase inputs are pulse and direction. If not specified, the inputs are pulse up and pulse down.
TE5312ICR_INC	Counter is manually incremented.
TE5312ICR_DEC	Counter is manually decremented.
TE5312ICR_ENABLE	Phase inputs are enabled.
TE5312ICR_GATE	The ABGT/RCTR input (which can be jumpered to the index input) is set up as the phase input enable / disable gate. If not specified, this input is set up as the counter external reset input.
TE5312ICR_LATCH	The LCTR / LLTC input (which can be jumpered to the index input) is set up as the external load command input for the Output Latch. If not specified, this input is set up as the external load command input for the counter.

For *OCCR,* OR any of the following constants together:

TE5312OCCR_BCD	If specified, the counter will be in BCD mode. If not specified, the counter will be in binary mode.
TE5312OCCR_NOCYCLE	If specified, the counter will not continually cycle.
TE5312OCCR_DIVIDE	Counter is set to divide-by-n mode.
TE5312OCCR_CLOCK	Sets counter to 24-hour clock mode.
TE5312OCCR_ACTIVE_LOW	Enables active LOW carry & borrow pulses on Cy & By outputs.
TE5312OCCR_TOGGLE	Enables carry & borrow toggle flip-flops on Cy & By outputs.

TE5312OCCR_ACTIVE_HIGH	Enables active HIGH carry & borrow pulses on Cy & By outputs.
TE5312OCCR_COMPARE	Enables compare pulses on Cy output and compare toggle flip-flop on By output.

The last four options are mutually exclusive.

For *QR*, one of the following constants can be specified:

TE5312QR_X1	1x Quadrature mode.
TE5312QR_X2	2x Quadrature mode.
TE5312QR_X4	4x Quadrature mode.

If zero is specified for QR, quadrature is disabled.

Return Code:

(0)	No error.
(-1)	Invalid axis number.

te5312InitSw

Initialize Software

Declarations:

C: `short te53l2InitSw(void);`

BASIC: `DECLARE FUNCTION te53l2InitSw%( )`

Pascal: `function  te53l2InitSw : integer;`

Description:

This routine initializes the 5312 software and must be called before calling any other driver routines.

Return Code:

The version number (four hex digits) of the 5312 driver is returned.

See Also:

te5312InitBoard

te5312InterruptHooks

Install Interrupt Hooks

Declarations:

C:
```
typedef void te5312HookType(short *psParm);
void InterruptHooks(te5312HookType *WrapAroundHook,
te5312HookType *IndexHook);
```

BASIC:
```
not supported.
```

Pascal:
```
te5312HookType = procedure(var psParm : integer);
procedure InterruptHooks(WrapAroundHook, IndexHook :
te53l2HookType);
```

Description:

Installs two routines as interrupt hooks to be called when the appropriate interrupt is generated. See the discussion in section 3 on interrupt handling.

Parameters:

WrapAroundHook	Routine to be called on a carry/borrow interrupt.
IndexHook	Routine to be called on an index interrupt.

te5312LoadCntr

Load Counter

Declarations:

C: `short te5312LoadCntr(short sAxisNum, long lValue);`

BASIC: `DECLARE FUNCTION te5312LoadCntr%(BYVAL AxisNum%, BYVAL Value&)`

Pascal: `function te5312LoadCntr(sAxisNum : integer; lValue : longint) : integer;`

Description:

This routine loads the specified value into the counter of the specified axis. It does this by first loading the value into the preset register and then commanding that the value of the preset register be transferred to the counter.

Parameters:

AxisNum Axis number (0 to 23) or global number (—1 to —6).
Value Value to be loaded into the counter.

Return Code:

(0) No error.

(—1) Invalid axis number.

See Also:

te5312Readcntr

te5312LoadPr

Load Preset Register

Declarations:

C: `short te5312LoadPr(short sAxisNum, long lValue);`

BASIC: `DECLARE FUNCTION te5312LoadPr%(BYVAL AxisNum%, BYVAL Value&)`

Pascal: `function te5312LoadPr(sAxisNum : integer; lValue : longint) : integer;`

Description:

This routine loads the specified value into the preset register of the specified axis.

Parameters:

AxisNum Axis number (0 to 23) or global number (-1 to -6).
Value Value to be loaded into the counter.

Return Code:

(0) No error.

(-1) Invalid axis number.

te5312ReadCntr

Read Counter

Declarations:

C: `long te53l2ReadCntr (short sAxisNum);`

BASIC: `DECLARE FUNCTION te5312ReadCntr&(BYVAL AxisNum%)`

Pascal: `function te5312ReadCntr(sAxisNum : integer) : longint;`

Description:

This routine reads and returns the current value of the counter.

Parameters:

AxisNum Axis number (0 to 23).

Return Code:

Current counter value.
(-1) Invalid axis number.

te5312ReadOL

Read Output Latch

Declarations:

C: `long te5312ReadOL(short sAxisNum).`

BASIC: `DECLARE FUNCTION te5312ReadOL&(BYVAL AxisNum%)`

Pascal: `function te5312ReadOL(sAxisNum : integer) : longint;`

Description:

This routine reads and returns the value of the output latch.

Parameters:

AxisNum Axis number (0 to 23).

Return Code:

Output latch value.
(-1) Invalid axis number.

te5312ReadSts

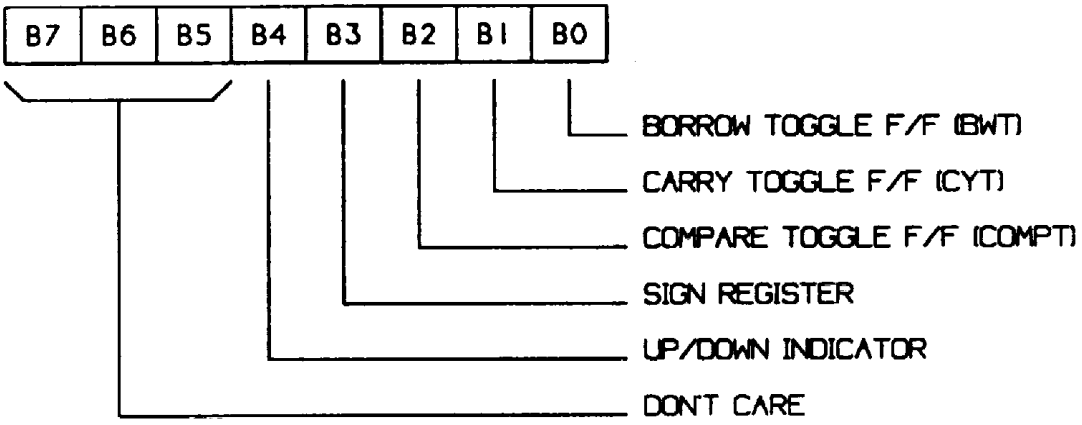
Read Status

C: `short te5312ReadSts(short sAxisNum);`

BASIC: `DECLARE FUNCTION te5312ReadSts%(BYVAL AxisNum%)`

Pascal: `function te5312ReadSts(sAxisNum : integer) : integer;`

Description:



This routine reads and returns the status register.

The bits of the status register formatted above can be masked out with the following constants:

TE5312STS_BORROW	Borrow Flag.
TE5312STS_CARRY	Carry Flag.
TE5312STS_COMPARE	Compare Flag.
TE5312STS_SIGN	Sign Flag.
TE5312STS_UP	Direction is up.

Parameters:

AxisNum Axis number (0 to 23).

Return Code:

Status Byte.

(—1) Invalid axis number.

te5312WrapAroundAlertOff

Disable Borrow / Carry Interrupt

Declarations:

C: `short te5312WrapAroundAlertOff(short sAxisNum);`

BASIC: `DECLARE FUNCTION te5312wrapAroundAlertOff%(BYVAL AxisNum%)`

Pascal: `function te5312wrapAroundAlertOff(sAxisNum : integer) : integer;`

Description:

This routine disables the carry or borrow interrupt for the specified axis.

Parameters:

AxisNum Axis number (0 to 23) or global number (-1 to -6).

Return Code:

(0) No error.

(-1) Invalid axis number.

See Also:

te5312WrapAroundAlertOn

te5312WrapAroundAlertOn

Enable Borrow/Carry Interrupt

Declarations:

C: `short te5312WrapAroundAlertOn(short sAxisNum);`

BASIC: `DECLARE FUNCTION te5312WrapAroundAlertOn%(BYVAL AxisNum%)`

Pascal: `function te53l2WrapAroundAlertOn(sAxisNum : integer) : integer;`

Description:

This routine enables the carry or borrow interrupt for the specified axis.

Parameters:

AxisNum Axis number (0 to 23) or global number (—1 to —6).

Return Code:

(0) No error.

(-1) Invalid axis number.

See Also:

te5312WrapAroundAlertOff

te5312WriteCmd

Write Command

Declarations:

C: `short te5312WriteCmd(short sAxisNum, short sCmd);`

BASIC: `DECLARE FUNCTION te5312WriteCmd%(BYVAL AxisNum%, BYVAL Command%)`

Pascal: `function te5312WriteCmd(sAxisNum, sCmd : integer) : integer;`

Description:

This routine writes the specified command to the specified axis.

Parameters:

AxisNum Axis number (0 to 23) or global number (-1 to -6).
Command Command to be written.

Command is constructed by ORing several constants together. It should always include one of the following four constants which identify the command register:

TE5312CMD_MCR	Master Control Register.
TE5312CMD_ICR	Input Control Register.
TE5312CMD_OCCR	Output / Counter Control Register.
TE5312CMD_QR	Quadrature Register.

If TE5312CMD_MCR is included, any of the following constants can also be ORed together:

TE5312MCR_ADDR_RST	Resets address counters.
TE5312MCR_CNT_OL	Loads Output Latch with counter value.
TE5312MCR_FLAG_RST	Zeroes counter, resets borrow/carry flags, sets sign flag.
TE5312MCR_PR_CNT	Loads counter with value of Preset Register.
TE5312MCR_CMP_RST	Resets compare flag.
TE5312MCR_MASTER_RST	Master reset.

If TE5312CMD_ICR is included, any of the following constants can also be ORed together:

TE5312lCR_DIR	Phase inputs are pulse and direction. If not specified, the inputs are pulse up/down.
TE5312ICR_INC	Counter manually incremented.
TE5312lCR_DEC	Counter manually decremented.
TE5312lCR_ENABLE	Phase inputs enabled.
TE5312lCR_GATE	ABGT/RCTR input (which can be jumpered to the index input) set up as phase input enable / disable gate. If not specified, input is setup as counter external reset input.
TE5312lCR_LATCH	The LCTR / LLTC input (which can be jumpered to the index input) set up as external load command input for Output Latch. If not specified, input set up as external load command input for counter.

If TE5312CMD_OCCR is included, any of the following constants can also be ORed together:

TE5312OCCR_BCD	Counter in BCD mode. If not specified, counter in binary mode.
TE5312OCCR_NOCYCLE	Counter not continual cycle.
TE5312OCCR_DIVIDE	Counter set to divide-by-n mode.
TE5312OCCR_CLOCK	Counter in 24-hour clock mode.
TE5312OCCR_ACTIVE_LOW	Enables active LOW carry/borrow pulses on Cy and By outputs.
TE5312OCCR_TOGGLE	Enables carry/borrow toggle flip-flops on Cy and By outputs.
TE5312OCCR_ACTIVE_HIGH	Enables active HIGH carry/borrow pulses on Cy and By outputs.
TE5312OCCR_COMPARE	Enables compare pulses on Cy output and compare toggle flip-flop on By output.

The last four options are mutually exclusive.

If TE5312CMD_QR is included, one of the following constants can also be ORed together:

TE5312QR_X1	1x Quadrature mode.
TE5312QR_X2	2x Quadrature mode.
TE5312QR_X4	4x Quadrature mode.

If none of the last three options is specified, quadrature is disabled.

Return Code:

(0) No error.

(-1) Invalid axis number.

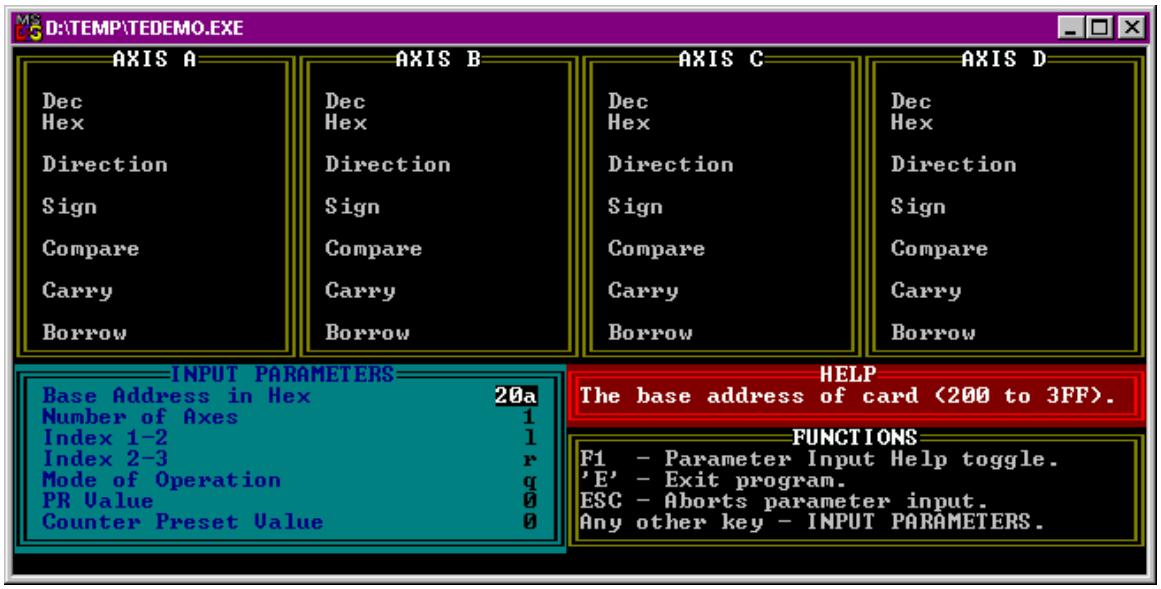
6 Demonstration Program

The 5312 includes a demonstration program designed to support the actual operation and general capabilities of the card. Each axis in the program can operate independently in any of the 5312 operating modes — quadrature, pulse/direction, and up/down counting. To run the program, insert the software diskette into drive A (or B), and then at the prompt type:

A:\EXE\TEDEMO <Enter> or

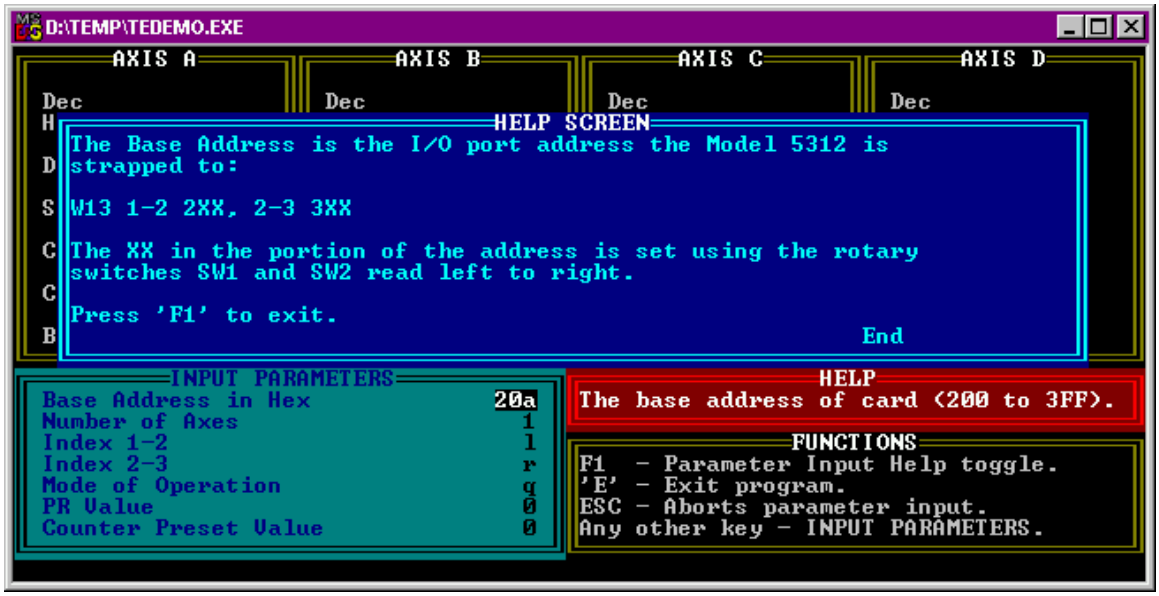
B:\EXE\TEDEMO <Enter>

After a moment, a brief message describing the program will appear on your screen. Pressing any key will continue the program, and the following screen will be displayed:



The program will ask you to define input parameters. Parameter choices can be seen in the window labeled HELP which appears to the right of the INPUT PARAMETERS window. For example, when choosing the number of axes, the HELP box will read the message “1- 4” denoting the number of axes that are possible to configure during the test.

For an explanation of the parameter to be set, press <Fl>. A help screen will appear containing a definition of the parameter. For example, pressing <Fl> when the cursor is at the PR Value will bring up the following screen:



After setting all the parameters and after setting the Counter Preset Value parameter, pressing <Enter> will cause values to be displayed in the boxes above. The actual number of boxes displaying data depends on the number of axes chosen.

The following is a brief description of each value:

Dec/Hex: Represents the count values. Dec represents the Decimal count; Hex represents the Hexadecimal count.

Direction: Direction to which the count is heading. The counter will either be heading up or down.

Sign: Indicates whether the counter has overflowed / underflowed. It will read plus when over flowed and minus when underflowed.

Compare/Carry/Borrow: These are toggles. Each one will read either HIGH or LOW.

Compare changes state every time the count equals the PR Value. Carry changes state every time there is an overflow. Borrow changes state every time there is an underflow. Refer to section 2 of this manual for further value descriptions in the Input Parameters.

To escape out to a DOS prompt, press <Esc> to exit the parameter list, and press <E> to exit TEDEMO.

7 Visual BASIC Demo Program

Overview

The Model 5312 also includes a 16-bit Windows Visual BASIC program that demonstrates the operation and general capabilities of the card. Each axis in the program can operate independently in any of the Model 5312 operating modes — quadrature, pulse/direction, and up/down counting.

The following appendix takes users through installation and operation of this demo. It assumes that they have the proper hardware configuration, including a properly configured 5312 plugged into the backplane of a Windows-equipped IBM compatible PC. Please see the *Model 5312 Technical Reference* for hardware installation procedures.

Software Installation

The 5312 Visual BASIC demo runs under Windows 3.1 or Windows 95. The common procedure is to run the file setup.exe from your 3½" floppy drive (a:\ or b:\). How you do this under the two environments depends on their respective user interfaces. The following is a detailed procedure for each environment.

Windows 3.1

Insert your 3½" 5312 Visual BASIC Demo diskette into your a: or b: floppy drive. In the Program Manager, pull down the File menu and select Run. You will get a dialogue box with one data field. Type a:\setup, or b:\setup and press <enter> or click OK. Windows 3.1 will run the installation procedure. Follow the steps as prompted by this procedure.

Windows 95

Insert your 3½" 5312 Visual BASIC Demo diskette into your a: or b: floppy drive. Click on the Start button at the lower left-hand part of your screen. Click on Run. Choose Browse and select Drive A: (or B: where appropriate). Click OK or press <enter> and Windows will run the installation procedure. Follow the steps as prompted by this procedure.

User's Guide

Double click on the VB5312 icon. The main user menu should appear as shown in figure 1.

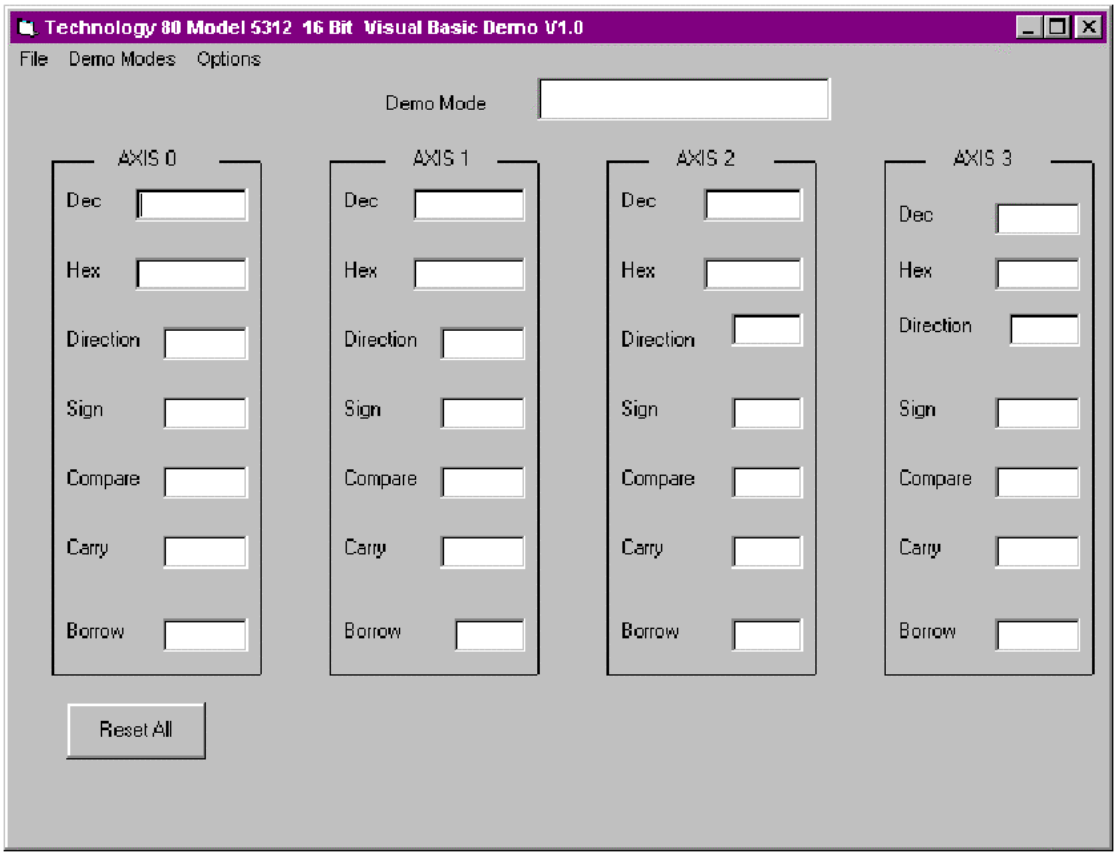


Figure 1: Main User Menu.

Note: if the board is not set at the correct address, you will get a "Hardware Initialization Error" during program startup. The default address is 300 hex. If another board in your backplane is configured at that address, you will have a conflict between that board and the Model 5312. To solve this conflict, follow the procedure below.

- Read the 5312 Technical Reference regarding changing board addresses. In this section, note how the switches on the board enable you to configure the board for a non-default address.

- In the menu depicted in Figure 1, pull down **Options**, choose **Board Address** and then select the address appropriate to the switch settings.

Demo Modes

The first logical operation is to select the demo mode you wish to operate in. To do this, pull down Demo Modes as shown in figure 2.

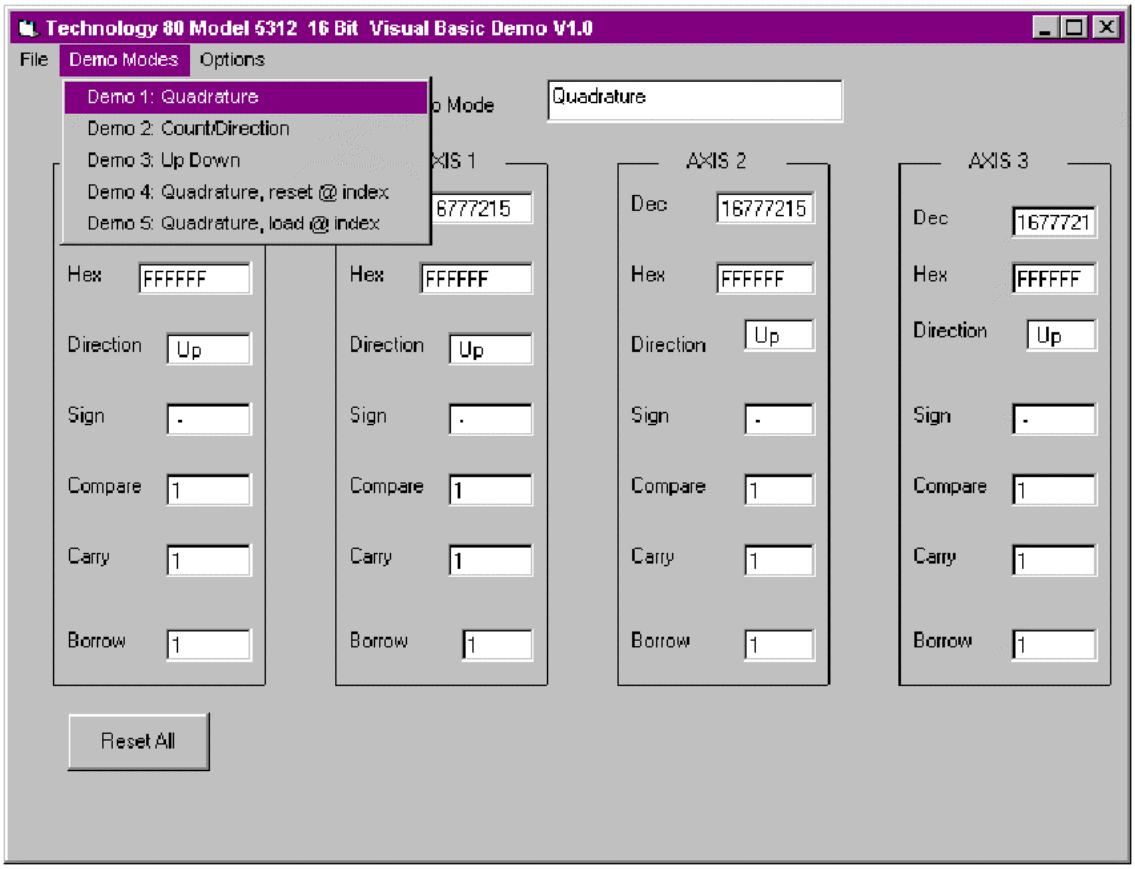


Figure 2 Main User Menu with **Demo Modes** Pulled Down

When you select a mode, a model dialogue box will appear. For example, if you select quadrature mode, the dialogue box shown in figures 3 will appear.

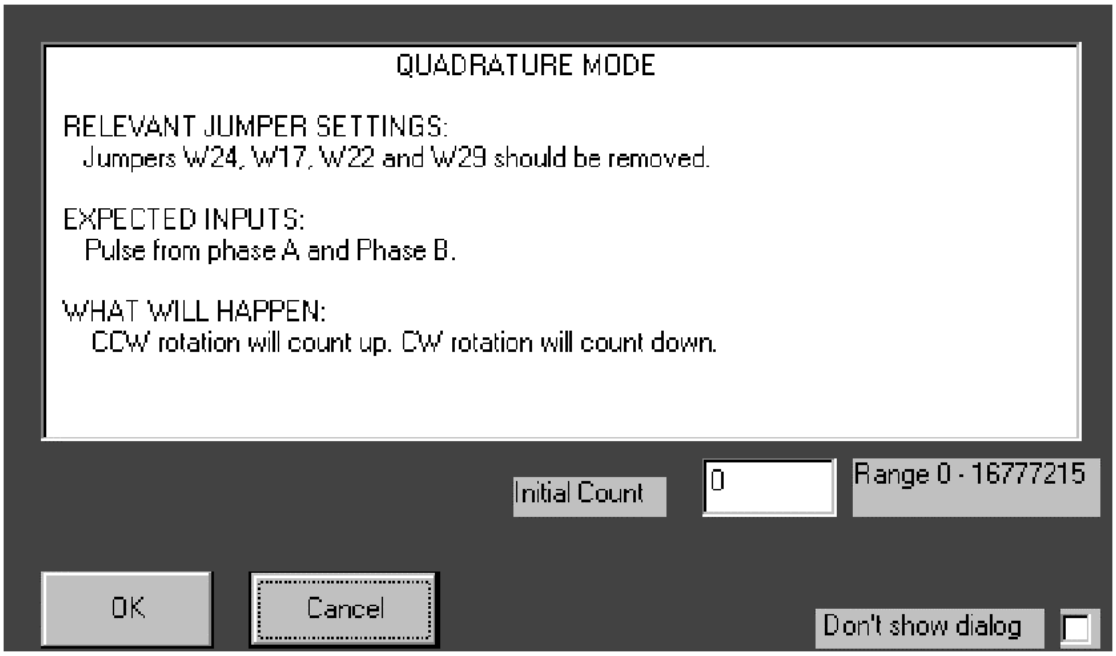


Figure 3 Quadrature Mode Dialogue Box

If you need to enter an initial count of other than 0, enter the desired count in the Initial Count data field. Then click OK or press <enter> to start the demo. The Reset All command button on the main user menu will reset all counters to zero. This is the default value. Dialogue boxes that appear when the user selects other modes are very similar to the above.

Developer's Guide

This demo was developed under Visual BASIC 4.0. For those unfamiliar with Visual BASIC, there are two primary types of modules developers create when they write a program — form and code modules. Form modules specify the form that GUI windows will take. Code modules are the guts of the program. They do everything from driving the system to enabling certain operations to take place within the form modules when users specify appropriate values. The architecture of any Visual BASIC program consists of a project file with some code and form modules in it.

The 5312 program architecture consists a project file (VB5312.VBP) of seven form modules and three code modules. The form modules enable the user to interact with the available GUI interfaces. The code modules run the board at a low level. We do not offer the source for these modules, so this section is for your information only.

Form Modules
The main form module is FORM12.FRM. It is comprised of menu items, text boxes and a command button. See Figure 4.

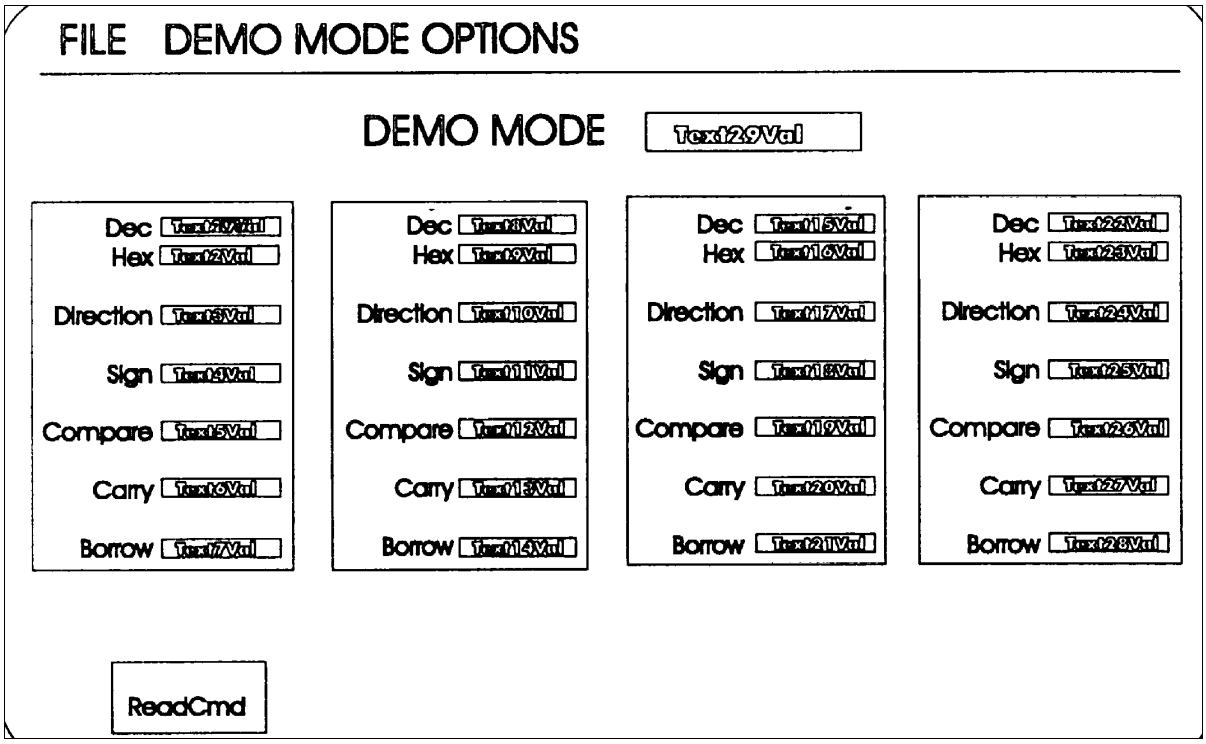


Figure 4 Primary form Module

The secondary form modules (DEMO1DLG.FRM through DEMO5DLG.FRM) are all quite similar. They comprise the dialogue boxes for the various demo modes. They consist of a large read-only text box that provides the information about the mode in question, a small changeable text box, two command buttons and a check box. See figure 5 (following figure) for a template.

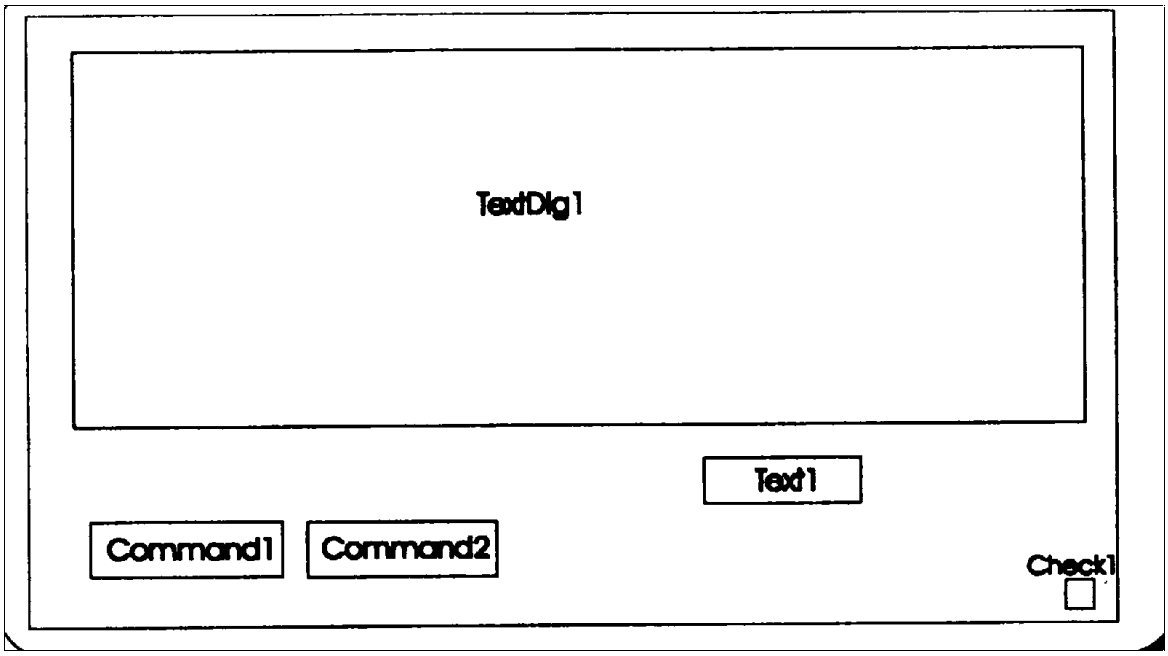


Figure 5 Template for form modules enabling demo-mode dialogue boxes

The last form module (ADDR.FRM) enables the user to enter board address values. It is not pictured here.

Code Modules

The three code modules in the 5312 demo (START.BAS, 5312H.BAS, AND DECL.BAS) drive the program. They are described in more detail below.

START. BAS

This code module is called on program startup. It runs through all the board initialization routines, including the init software, init board and init global variables.

5312H.BAS

This module contains board register constants, dynamic linking library (DLL) function declarations and DLL prototypes.

DECL.BAS

This code module contains program constants, user-defined types and global variables.

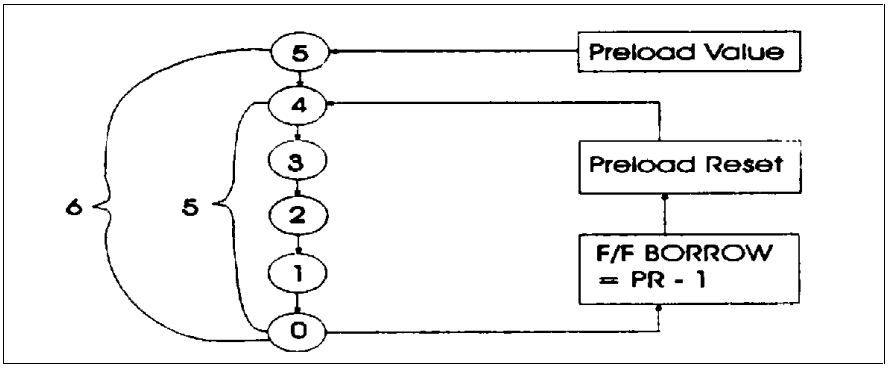
8 LSI Chip Applications Note

Introduction

The following is an application note for developers of products with Model 5312. Those who preset their counter values relatively high (thousands of counts, typical) possibly won't notice this problem in most applications. In fact, with thousands of units in the field, we just recently discovered the bug.

Problem Definition

The LSI 7166 chipset onboard these models exhibits a minor logic anomaly. The problem is most noticeable on systems using relatively low preloaded count values in the 7166. For example, if you preset your counter at 5, the first count loop is decremented as 5, 4, 3, 2, 1, 0 then carries as the count rolls on to ffffff. Thus, the count loop consists of 6 states. On the second and all subsequent count loops, as the counter is decremented through zero, a borrow or carry condition occurs. The counter-is then reset and preloaded with a value equal to the original value minus 1, i.e. [preload - borrow] or [5-1 = 4 (see figure D.1).



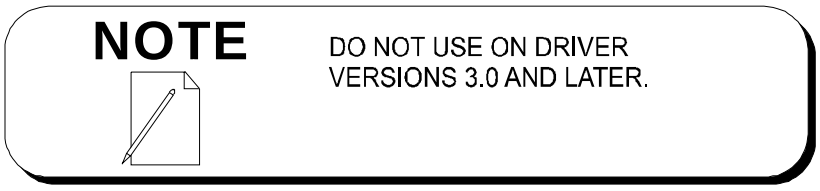
LSI Chipset Counter Problem
This makes the problem associated with low counts per index manifest.

1.2 Problem Solution

The solution to this problem is to preload the counter to '1', load the preset register to '5', do a manual decrement twice. This will cause the counter to be reloaded with the correct value [preload - borrow]. You can perform this before the main counter control loop. To solve the problem at the register level, perform the increment/decrement at bits 2 and 1 of the Input Control Register during the initialization and configuration stage. (See the manual.) To do this with the software drivers, follow the source below (in C). In this example, it is assumed that the axis is 0 and the desired counts per update cycle is 5. Of course, you will need to pass axis and desired counts per cycle parameters appropriate to your application.

```
/ * LOU COUNTS PER INDEX INITIALIZATION ROUTINE * /
te5312DisableIRQ( ); //——Mask Interrupt
te5312LoadPr (Axis0, 1); //——Preload Counter to 1
te5312WriteCmd (Axis0, TE5312CR_DEC); II—— Decrement Counter te5312WriteCmd
(Axis0, TE5312CR_DEC); II—— Twice
te5312LoadPr (Axis0, 5); //——Reload Desired Cycle Counter
/ * Ready to Go * /
```

9 Using the Model 5312 Windows NT Drivers



Installation

The Model 5312 Windows NT drivers diskette contains all the necessary files for complete NT operation. This operation is supported under Windows NT 4.0.

The Installation Program can be used if Windows NT is installed in your default directories. If NT is installed in another directory, the files will have to be copied manually into the correct directories. Please see the Manual Installation section below for complete instructions.

Normal Installation

- Make sure that you are logged into your NT workstation and that you have administrative user rights.
- Insert the driver diskette. Click Start on your NT desktop.
- Go to Programs. Click on MS -DOS. Type the drive letter of your diskette.
- Type "install" and press Enter. Kernel and user mode drivers will be copied to the appropriate directories, and an appropriate key will be added to the registry.
- Remove the diskette. Restart the computer.
- `T5312NT.sys` is shipped as a manual start driver. To start the driver, go to the MS-DOS prompt and type "net start T5312NT." The words, "Service started successfully," should appear. If any other message appears, please see the Troubleshooting Section below.
- Upon a successful start of the 5312NT kernel mode driver, the Parameters - Start option may be changed from 3 to 2 in the registry for automatic start operation. See the following page for instructions on how to modify parameters in the registry.

Manual Installation

- Make sure that you are logged into your NT workstation and that you have administrative user rights.
- Insert the driver diskette. Click Start on your NT desktop.
- Copy `TE5312NT.dll` into your WINNTDIR\SYSTEM32 directory.
- Copy `T5312NT.sys` into your WINNTDIR\SYSTEM32\DRIVERS directory.
- Type “DINI T5312NT.ini”
- Remove the diskette. Restart the computer.

- To start the driver, go to the MS - DOS prompt and type "net start T5312NT" The words, "Service started successfully", should appear. If any other message occurs, please see the Troubleshooting Section below.
- Upon a successful start of the 5312NT kernel mode driver, the *Parameters - Start* option may be changed from 2 to 3 in the registry for automatic start operation. Please see the section below for instructions on how to modify parameters in the registry.

Troubleshooting

If you get, "System error 87," the parameter is incorrect. This means you have a conflicting address and interrupt number, and that another NT driver has possession of one or both of the parameters.

To correct this, run WINMSD. Look under Resources and I/0 Port for information about which IRQs and lO Ports are assigned. Disable the conflicting driver or change the conflicting drivers' IRQ and/or port I/0, or change the 5312 NT parameters as described below

If you get, "System error 32," the system cannot find the file specified. This will happen if you did not copy over the kernel mode driver or if you have it in the wrong directory.

If you get, "The service name is invalid," the registry key for the driver is not set up. Please see "Modifying the Registry" below for instructions on how to verify that the key is present.

If you are still having problems, please contact ACS-Tech 80 at the numbers shown at the beginning of this document for assistance. Our applications engineers will be happy to work with you.

Modifying the Registry

To modify the registry,

- Click Start on the Windows NT Desktop. Click "Run." Type "Regedit." Click OK.
- Click HKEY_LOCAL_MACHINE.
- Click System. Click Current Control Set. Click Services. Click T5312NT.

To modify the start mode, click the T5312NT folder. To change either the address or the IRQ number, select the Parameters folder.

- To change the value, click the name of the value you wish to change. Then click Edit. Click Modify.
- Enter the new value and click OK. Close the registry editor and restart the computer.

Driver Overview

Using the ACS-Tech80 Model 5312 under Windows, NT requires the use of both a kernel mode driver and a user mode driver. The kernel mode driver provides I/O communication between the user mode driver and the 5312 board. The user mode driver provides communication between the application and the kernel mode driver.

NT Functions

The following functions have been added for NT operation:

```
OpenFileLink
CloseFileLink
InitIntr
WaitForInt
IntRead
DisableIntr
```

Operations

Basic NT operation requires the addition of the `OpenFileLink` and `CloseFileLink` functions.

Visual Basic Operation

To use Visual Basic:

- Place the `OpenFileLink` call before calling any other Model 5312 function calls, other than `InitSw`. `OpenFileLink` can be put in *Start-up Main* or in an *OK* command button in a dialogue box.
- Place `CloseFileLink` in the *Main Window Unload Process* of the main window form. If you do not use `CloseFileLink`, you will lose file handles, and at some point, you may not be able to start up the application.

Visual C++ Operation

When using Visual C++:

- Place the `OpenFileLink` call before calling any other Model 5312 functions, other than `InitSw`. `OpenFileLink` can be put in either a constructor in or a command button.
- Place `Close File Link` in a destructor. If you do not use `CloseFileLink`, you will lose file handles, and at some point, you may not be able to start up the application.

Using Interrupts

Interrupts are only supported under Visual C++.

Under VC++, the only interrupt-related functions are:

```
InitIntr
WaitForInt
IntRead
UserInputAlertOn
UserInputAlertOff
```

To set up for an interrupt operation, follow these directions:

To define a thread message, type:

```
@define WM_ON_THREAD (WM_USER + 1)
```

To create an instance of the int info structure, type:

```
INTSTATUS *PINTSTATUS
```

To declare and instance of CW.WThread:

```
CWinThread* pThread;
```

To prototype a function to handle the worker thread:

```
static UINT Poll(LPVOID pParm);
```

To create a corresponding function to process the worker thread:

```
UNIT userWindow::Poll (LPVOID pParm)
{
      threadActive = TRUE;        //Global variable to keep track
                                  //of thread status
      WaitForInt();               //Wait for the interrupt
                                  //Post the Windows message.
      ::PostMessage((HWND) pParm,WM_ON_THREAD,0,0);
      ThreadActive = FALSE;
      return (0);                 //End of thread.
}
```

To prototype and create a function for determining the cause of the interrupt:

```
afx_msg long ProcInt (UINT wParam, Long 1Param)


long UserWindow:: ProcInt (UINT wParam, Long 1 Param)+
PINTSTATUS = IntRead();


//Put the next line in to report all interrupt occurrences.
//pThread = AfxBeginThread(Poll, (LPVOID) GetSafeHwnd());

if((PINTSTATUS->AxisNum == 0) && (PINTSTATUS -> CauseOfIntr == WRAP))
{
     MessageBox("Axis 1 Wrap Intr" "InterruptP"'MB_OK);
}
else if ((PINTSTATUS->AXISNUM == AXIS1) &&
        (PINTSTATUS -CauseOfIntr == INDEX))
     }
MessageBox("Axis 1 INDEX Intr","InterruptP,MB_OK):
}
```

To map the `ProcInt` function to the worker thread message:

```
     ON_MESSAGE(WM_ON_THREAD, ProcInt)
```

To start the worker thread in an applications command button:

```
void UserWindow::RestartAll ()
{

(void)     te5312LoadCntr(0.0);
(void)     te5312WriteCmd(-1 TE5312MC_FLAG_RST);
(VOID)     TE5312WriteCmd(-1, TE5312MCR_CMP_RST);

if (IntrEnableFlag)
{
     Thread = AfxBeginThread (Pol,1, (LPVOID) GetSafeHwnd ();
}
```

Worker Thread Management

It is important to keep track of when a worker thread is started and when it is stopped.

Interrupting Axis Numbers

The interrupting axis numbers are directly reported. Interrupt causes are:

0 = WRAP

1 = INDEX.

DRIVER ROUTINE REFERENCES

OpenFileLink

Declarations:

C: `int OpenFileLink(void)`

Visual Basic:
```
declare function OpenFileLink Lib_ "te5312NT" ()
                                                As Integer
```

Description:

Opens a symbolic link between user and kernel mode drivers.

Parameters:

None

Return Code:

Returns 0 upon success.
Returns -1 upon failure.

See Also:

CloseFileLink

CloseFileLink

Declarations:

C:
```
void CloseFileLink (void)
```

Visual Basic:
```
declare function CloseFileLink Lib_ "te5312NT" ( )
                                                    As Integer
```

Description:

Closes symbolic link between user and kernel mode drivers

Parameters:

None

Return Code:

Returns 0 upon success.
Returns -1 upon failure.

See Also:

OpenFileLink

InitIntr

Declarations:

C:: `int InitIntr (void)`

Description:

Retrieves an event object's handle.

Parameters:

None

Return Code:

Returns 0 upon success.
Returns - 1 upon failure.

See Also:

WaitForInt

WaitForInt

Declarations:

C: `void WaitForInt (void)`

Description:

Waits for a system event.

Parameters:

None

Return Code:

None.

See Also:

InitInt, DisableIntr

IntRead

Declarations:

C: `INTSTATUS* IntRead (void)`

Description:

Allows access to data members of structure INTSTATUS.
These data members contain information about which axis incurred.
The interrupt and its cause (such as end of motion or user input.)

Parameters:

None

Return Code:

Returns a pointer to structure INTSTATUS.
Returns - 1 upon failure.

See Also:

file te5312w.h

Structure Members:

AxisNum
CauseOfInter

DisableIntr

Declarations:

C: `voidDisableIntr (void)`

Description:

Closes an event object's handle

Parameters:

None

Return Code:

None

See Also:

InitIntr

10 Using the Model 5312 32-bit Windows Drivers

Installation

The Model 5312 Windows 32-bit NT drivers diskette contains driver files that will allow you to use all 5312 functions *except* those relating to interrupts.

To begin installation, copy filesT5312_95.dll and T5312_95.vxd into the \WINDOWS_SYSTEM directory. For Visual Basic operation, no other files are required.

Visual C++

For Visual C++, GUI applications, copy file T5312_95.lib into your Application Development directory. You must also include T5312_95.lib in your application project.

For Visual C++ Console applications, copy files T53_98.lib and T5312_95.exp into your Application Development directory. Include T5312_95.lib in your application project.

DRIVER ROUTINE REFERENCES

OpenTheDriver

Declarations:

C: `void OpenTheDriver (void)`

Description:

Opens a symbolic link between user and kernel mode driver.

Parameters:

None

Return Code:

Returns 0 upon success.
Returns - 1 upon failure.

See Also:

CloseTheDriver

CloseTheDriver

Declarations:

C: `void CloseTheDriver (void)`

Description:

Releases the file handle that is used by **OpenTheDriver**

Parameters:

None

Return Code:

None

See Also:

OpenTheDriver

11 Manual Addendum

Please incorporate the following into your ACS-Tech80 Software Guide under "Troubleshooting."

Linker Errors for C++ Users

"Unresolved external" error
An unresolved external linker error may be caused by one of two situations.

1. The user does not have the corresponding .LIB file for the .DLL used in the application. For example, if a user is using TE5638W.DLL, he or she must have TE5638W.LIB in the VC++ project. This is because the .LIB file is what tells the linker ***where*** to look for functions.

The .LIB file is usually kept in the working directory, although it can be anywhere in the path. The .DLL file is ***usually*** kept in the \WINDOWS\SYSTEM directory, although it to may be anywhere in the path.

2. The customer has not disabled name mangling (Borland) or name coding (Microsoft) for the header file (.H). To disable name mangling or name coding: (using the te5650 as an example.)

```
extern "C"
{
   #include "te5650.h"
}
```

This must be done because the functions are prototyped in an .H file as opened to a .CPP file. Because they are in a non-CPP file, they are considered external functions. If the functions were prototyped in both a .CPP file and a .H file or a .C file, the linker wouldn't know which file to use. This means that any functions outside of a .CPP file gets name mangled unless the extern C directive otherwise directs it. This feature of the C++ language is called type safe linkage.

If the extern C directive is not used, the project will compile correctly, but errors on the link occur because the function names in the .OBJ file created by the compilation are mangled. These function names do not match the unmangled function names in the .LIB file.

"Corrupted or missing file" error

The project builds up but the "corrupted or missing file" error occurs when the application runs.

This error is rare. It is caused when the user attempts to access a 16-bit DLL from a 32-bit application or vice versa. Since the data word sizes are different, it is unable to accurately read the file.