Software

Model 5312B
Software Developer's Guide V1.0

KEITHLEY

80370 Rev. B / 7-98

WARRANTY

Hardware

Keithley Instruments, Inc. warrants that, for a period of one (1) year from the date of shipment (3 years for Models 2000, 2001, 2002, and 2010), the Keithley Hardware product will be free from defects in materials or workmanship. This warranty will be honored provided the defect has not been caused by use of the Keithley Hardware not in accordance with the instructions for the product. This warranty shall be null and void upon: (1) any modification of Keithley Hardware that is made by other than Keithley and not approved in writing by Keithley or (2) operation of the Keithley Hardware outside of the environmental specifications therefore.

Upon receiving notification of a defect in the Keithley Hardware during the warranty period, Keithley will, at its option, either repair or replace such Keithley Hardware. During the first ninety days of the warranty period, Keithley will, at its option, supply the necessary on site labor to return the product to the condition prior to the notification of a defect. Failure to notify Keithley of a defect during the warranty shall relieve Keithley of its obligations and liabilities under this warranty.

Other Hardware

The portion of the product that is not manufactured by Keithley (Other Hardware) shall not be covered by this warranty, and Keithley shall have no duty of obligation to enforce any manufacturers' warranties on behalf of the customer. On those other manufacturers' products that Keithley purchases for resale, Keithley shall have no duty of obligation to enforce any manufacturers' warranties on behalf of the customer.

Software

Keithley warrants that for a period of one (1) year from date of shipment, the Keithley produced portion of the software or firmware (Keithley Software) will conform in all material respects with the published specifications provided such Keithley Software is used on the product for which it is intended and otherwise in accordance with the instructions therefore. Keithley does not warrant that operation of the Keithley Software will be uninterrupted or error-free and/or that the Keithley Software will be adequate for the customer's intended application and/or use. This warranty shall be null and void upon any modification of the Keithley Software that is made by other than Keithley and not approved in writing by Keithley.

If Keithley receives notification of a Keithley Software nonconformity that is covered by this warranty during the warranty period, Keithley will review the conditions described in such notice. Such notice must state the published specification(s) to which the Keithley Software fails to conform and the manner in which the Keithley Software fails to conform to such published specification(s) with sufficient specificity to permit Keithley to correct such nonconformity. If Keithley determines that the Keithley Software does not conform with the published specifications, Keithley will, at its option, provide either the programming services necessary to correct such nonconformity or develop a program change to bypass such nonconformity in the Keithley Software. Failure to notify Keithley of a nonconformity during the warranty shall relieve Keithley of its obligations and liabilities under this warranty.

Other Software

OEM software that is not produced by Keithley (Other Software) shall not be covered by this warranty, and Keithley shall have no duty or obligation to enforce any OEM's warranties on behalf of the customer.

Other Items

Keithley warrants the following items for 90 days from the date of shipment: probes, cables, rechargeable batteries, diskettes, and documentation.

Items not Covered under Warranty

This warranty does not apply to fuses, non-rechargeable batteries, damage from battery leakage, or problems arising from normal wear or failure to follow instructions.

Limitation of Warranty

This warranty does not apply to defects resulting from product modification made by Purchaser without Keithley's express written consent, or by misuse of any product or part.

Disclaimer of Warranties

EXCEPT FOR THE EXPRESS WARRANTIES ABOVE KEITHLEY DISCLAIMS ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. KEITHLEY DISCLAIMS ALL WARRANTIES WITH RESPECT TO THE OTHER HARDWARE AND OTHER SOFTWARE.

Limitation of Liability

KEITHLEY INSTRUMENTS SHALL IN NO EVENT, REGARDLESS OF CAUSE, ASSUME RESPONSIBILITY FOR OR BE LIABLE FOR: (1) ECONOMICAL, INCIDENTAL, CONSEQUENTIAL, INDIRECT, SPECIAL, PUNITIVE OR EXEMPLARY DAMAGES, WHETHER CLAIMED UNDER CONTRACT, TORT OR ANY OTHER LEGAL THEORY, (2) LOSS OF OR DAMAGE TO THE CUSTOMER'S DATA OR PROGRAMMING, OR (3) PENALTIES OR PENALTY CLAUSES OF ANY DESCRIPTION OR INDEMNIFICATION OF THE CUSTOMER OR OTHERS FOR COSTS, DAMAGES, OR EXPENSES RELATED TO THE GOODS OR SERVICES PROVIDED UNDER THIS WARRANTY.

KEITHLEY

Keithley Instruments, Inc. • 28775 Aurora Road • Cleveland, OH 44139 • 440-248-0400 • Fax: 440-248-6168 • http://www.keithley.com

CHINA: **Keithley Instruments China** • Yuan Chen Xin Building, Room 705 • 12 Yumin Road, Dewai, Madian • Beijing 100029 • 8610-62022886 • Fax: 8610-62022892
FRANCE: **Keithley Instruments SARL** • BP 60 • 3 Allée des Garays • 91122 Palaiseau Cédex • 33-1-60-11-51-55 • Fax: 33-1-60-11-77-26
GERMANY: **Keithley Instruments GmbH** • Landsberger Strasse 65 • D-82110 Germering, Munich • 49-89-8493070 • Fax: 49-89-84930759
GREAT BRITAIN: **Keithley Instruments, Ltd.** • The Minster • 58 Portman Road • Reading, Berkshire RG30 1EA • 44-1189-596469 • Fax: 44-1189-575666
ITALY: **Keithley Instruments SRL** • Viale S. Gimignano 38 • 20146 Milano • 39-2-48303008 • Fax: 39-2-48302274
NETHERLANDS: **Keithley Instruments BV** • Avelingen West 49 • 4202 MS Gorinchem • 31-(0)183-635333 • Fax: 31-(0)183-630821
SWITZERLAND: **Keithley Instruments SA** • Kriesbachstrasse 4 • 8600 Dübendorf • 41-1-8219444 • Fax: 41-1-8203081
TAIWAN: **Keithley Instruments Taiwan** • 1FL., 85 Po Ai Street • Hsinchu, Taiwan • 886-3-572-9077 • Fax: 886-3-572-9031

Model 5312B
Software Developer's Guide V1.0

Cleveland, Ohio, U.S.A.
Second Printing, July 1998
Document Number: 80370 Rev. B

Manual Print History

The print history shown below lists the printing dates of all Revisions and Addenda created for this manual. The Revision Level letter increases alphabetically as the manual undergoes subsequent updates. Addenda, which are released between Revisions, contain important change information that the user should incorporate immediately into the manual. Addenda are numbered sequentially. When a new Revision is created, all Addenda associated with the previous Revision of the manual are incorporated into the new Revision of the manual. Each new Revision includes a revised copy of this print history page.

Revision A (Document Number 80370) ..... August 1996
Revision B (Document Number 80370) ..... July 1998

About this manual

Quality control

Keithley Instruments 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 fill out and mail the reader response card at the end of this manual (postage is prepaid).

Conventions

Procedural

Keithley Instruments uses various conventions throughout this manual. 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 **A warning is used to indicate that an action must be done with great care. Otherwise, personal injury may result.**

CAUTION **A caution is used to indicate that an action may cause minor equipment damage or the loss of data if not performed carefully.**

NOTE *A note is used to indicate important information needed to perform an action or information that is nice-to-know.*

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

Notational

A forward slash (/) 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 lower 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 font when they appear outside of function declarations, e.g., ReadStatus.

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.

Table of Contents

A Driver Routine Descriptions

B Demonstration Program

C Visual BASIC Demonstration Program

D LSI Chip Applications Note

List of Illustrations

A Driver Routine Descriptions

B Demonstration Program

C Visual BASIC Demonstration Program

D LSI Chip Application Note

List of Tables

4 Alphabetical Routine Summary

1
Programming Overview

Installing the 5312 software

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

```
install c:\5312
```

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.

A BASIC subdirectory, a C subdirectory, and a Pascal subdirectory will be created off of the directory you specify, and you may delete any unneeded subdirectories to save disk space.

Compiling and linking

The following paragraphs describe how to compile a program using the 5312 driver with the various supported compilers. It is assumed the source file is named DEMO.C for C, DEMO.BAS for BASIC, and DEMO.PAS for Pascal.

Microsoft C or Microsoft QuickC

To compile and link on the command line, enter the following:

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

where x is:

s	small model,
m	medium model,
c	compact model,
l	large model

Turn stack checking off with the /Gs switch (option) if you use interrupts. For CodeView compatibility, include the /Zi switch.

To use the 5312 driver in the QuickC environment, perform 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.

Borland or Turbo C/C++

To compile and link on the command line, enter the following:

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

where x is:

s	small model
m	medium model
c	compact model
l	large model

For Turbo Debugger compatibility, include the -v option.

To use the 5312 driver in the Borland environment, perform 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). If you use interrupts, turn stack-checking off.

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, enter the following:

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

If you use the QuickBASIC environment, you first have to run the batch file QLB5312.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: te5312qb.qlb and te5312qb.lib. Library te5312qb.qlb is a quick library for use in the QuickBASIC environment and te5312qb.lib is the command line equivalent. Therefore, you will develop your program with te5312qb.qlb and then in the final compilation, link with te5312qb.lib.

To use the 5312 driver in the QuickBASIC environment, enter the following:

```
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 te5312qb.lib are similar but not identical. Library te5312b.lib calls two routines not contained in the library itself: te5312IndexAlert and te5312WrapAroundAlert. These two routines must 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 cannot be converted into a quick library.

The library 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 te5312qb.qlb. A program developed in the QuickBASIC environment using te5312qb.qlb can be compiled on the command line and linked with te5312qb.lib without modifying the source code. See the information on using interrupts with BASIC.

Borland Turbo Pascal

To compile and link on the command line, enter the following:

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

If you use interrupts, be sure to turn stack-checking off. Turn off stack-checking by including /$S on the command line as shown or by including the line {$S-} in the program source code.

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

To use the 5312 driver in the Turbo Pascal environment, enter the following:

```
turbo demo
```

The source file must include the line: uses te5312p;. If you use interrupts, be sure to turn stack checking off. Turn off stack-checking through the Options/Compiler menu or by including the line {$S-} in the program.

Programming fundamentals

To quickly write simple applications for the 5312, follow the structure of the example programs provided in section 2. For C, include the file te5312.h. For BASIC, include the TE5312.BAS file. For Pascal, always specify the te5312p unit.

Call te5312InitSw first to initialize the software. Then call te5312InitBoard once for every 5312 board in the system. To use the other driver routines, you must be familiar with the concept of board, axis, and global numbers.

Each board in the system will be sequentially assigned a number from 0 to 5, called the board number, used to identify the board in calls to other routines. Each time te5312InitBoard is called, another board number is assigned. If only one board is installed in the system, calling te5312InitBoard once assigns a board number of zero.

Likewise, each axis in the system will be sequentially assigned an axis number from 0 to 23, used to identify a particular axis in calls to other routines. Each time te5312InitBoard is called, three more axis numbers are assigned.

Each board is also assigned a global number from –1 down to a possible –6. Global numbers can be used in place of axis numbers in routines that write to an encoder. In these cases all the encoders on the corresponding board will be written at the same time.

2
Example Programs

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, wIndexB;

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
  te5312InitSw();

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

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

  // initialize interrupts
  te5312InterruptHooks(te5312WrapAroundAlert, te5312IndexAlert);
  te5312EnableIRQ(BOARD, sIRQNum);
  te5312IndexAlertOn(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 = te5312ReadCntr(AXIS_A); lCntB = te5312ReadCntr(AXIS_B);
    sStatA = te5312ReadSts(AXIS_A); sStatB = te5312ReadSts(AXIS_B);
    printf("\r%8ld   %2X     %5u      %5u     ", lCntA, sStatA,
      wIndexA, wCarryA);
    printf("%8ld   %2X     %5u      %5u", lCntB, 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){
  case AXIS_A: wIndexA++; break;
  case AXIS_B: wIndexB++; break;
  }
}
```

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% = te5312IndexAlertOn(-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 exiting 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
END SUB
```

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 := te5312IndexAlertOn(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 := te5312ReadCntr(AXIS_A);
    lCntB := te5312ReadCntr(AXIS_B);
    sStatA := te5312ReadSts(AXIS_A);
    sStatB := 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

Introduction

The 5312 driver 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.

NOTE Interrupt Request (IRQ) address variables must be declared GLOBAL.

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.

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). Turn off stack-checking for the interrupt hook functions and any routines they call.

Interrupts in BASIC

The example program given in Section 2 shows how interrupts are used. You must provide two routines: te5312WrapAroundAlert and te5312IndexAlert**.**

You can use interrupts in the QuickBASIC environment, but the interrupt handling routines must be in the Quick Library te5312qb.qlb. To do this, use the INTR5312.BAS file to write your interrupt hook routines. Run the batch file QLB5312.BAT to compile INTR5312.BAS and add it to the libraries, te5312qb.qlb and te5312qb.lib. The library te5312qb.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.

General notes on using interrupts

There are some important points to be aware of when using interrupts:

1. **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. Do 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. When the main program sees the flag set, it will print the message.
2. **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

Introduction

The 5312 driver software consists of the following routines. A more complete description of each is given in Appendix A.

Table 4-1
Notational conventions

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

Routines

te5312DisableIRQ()	Restores old interrupt vectors and disables PC IRQ lines.
te5312EnableIRQ(wBoardNum, sIRQLevel)	Sets up interrupt vector and enables PC IRQ line on the bus.
te5312IndexAlertOff(sAxisNum)	Disables index interrupts.
te5312IndexAlertOn(wAxisNum)	Enables index interrupts.
te5312InitBoard(wBoardAddr, wNumAxes)	Initializes 5312 board.
te5312InitEncoder(sAxisNum, sMCR, sICR, sOCCR, sQR)	Initializes encoder.
te5312InitSw()	Initializes software.
te5312InterruptHooks(*WrapAroundHook, *IndexHook)	Defines hooks to user functions called on interrupts (not usable in BASIC).
te5312LoadCntr(sAxisNum, lValue)	Loads encoder counter.
te5312LoadPr(sAxisNum, lValue)	Loads encoder preset register.
te5312ReadCntr(sAxisNum)	Reads encoder counter.
te5312ReadOL(sAxisNum)	Reads encoder output latch.
te5312ReadSts(sAxisNum)	Reads encoder status.
te5312WrapAroundAlertOff(sAxisNum)	Disables borrow/carry interrupt.
te5312WrapAroundAlertOn(sAxisNum)	Enables borrow/carry interrupt.
te5312WriteCmd(sAxisNum, sCmd)	Writes encoder command.

A
Driver Routine Descriptions

Appendix A

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 lower-case prefix:

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

For instance, sAxisNum indicates that this variable is an 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, te5312InitSw.

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 te5312DisableIRQ(void);
```
BASIC:
```
DECLARE FUNCTION te5312DisableIRQ%()
```
Pascal:
```
function te5312DisableIRQ : integer;
```

Description: This routine masks the IRQ lines selected with te5312EnableIRQ calls and restores the corresponding interrupt vectors to their original values. If te5312EnableIRQ has been called at least once, call te5312DisableIRQ before exiting from 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 selected interrupt vector to point to the driver interrupt handler for the specified board. It also saves the old vector and 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: te5312DisableIRQ

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 te5312InitSw first to initialize the software, then call te5312InitBoard 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.

te5312InitBoard initializes the board interrupt controller and each encoder. For each encoder, te5312InitBoard 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.

Override this call by calling te5312InitEncoder.

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, sICR, 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 Register 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 and 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	1*x* Quadrature mode.
TE5312QR_X2	2*x* Quadrature mode.
TE5312QR_X4	4*x* 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 te5312InitSw(void);
```
BASIC:
```
DECLARE FUNCTION te5312InitSw%()
```
Pascal:
```
function te5312InitSw : integer;
```

Description: This routine initializes the 5312 software. Call this routine before calling any other driver routine in your program.

Return Code: The version number (4 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 available).

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

Description: Installs two routines as interrupt hooks to be called when the appropriate interrupt is generated. See the discussion in Section 3 for more 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 te5312ReadCntr(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

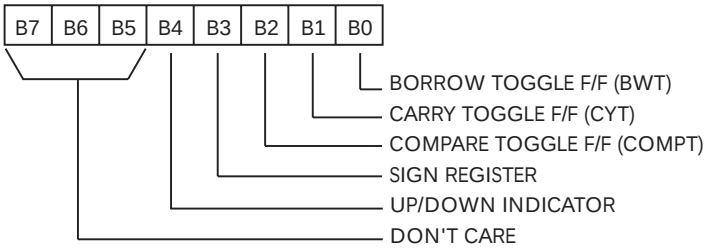
Read status

Declarations:

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.

Figure A-1
Read status



The bits of the status register formatted in Figure A-1 can be masked out using 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 te5312WrapAroundAlertOn (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:

TE5312ICR_DIR	Phase inputs are pulse and direction. If not specified, the inputs are pulse up/down.
TE5312ICR_INC	Counter manually incremented.
TE5312ICR_DEC	Counter manually decremented.
TE5312ICR_ENABLE	Phase inputs enabled.
TE5312ICR_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 set up as counter external reset input.
TE5312ICR_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	1*x* Quadrature mode
TE5312QR_X2	2*x* Quadrature mode
TE5312QR_X4	4*x* Quadrature mode

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

Return Code:

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

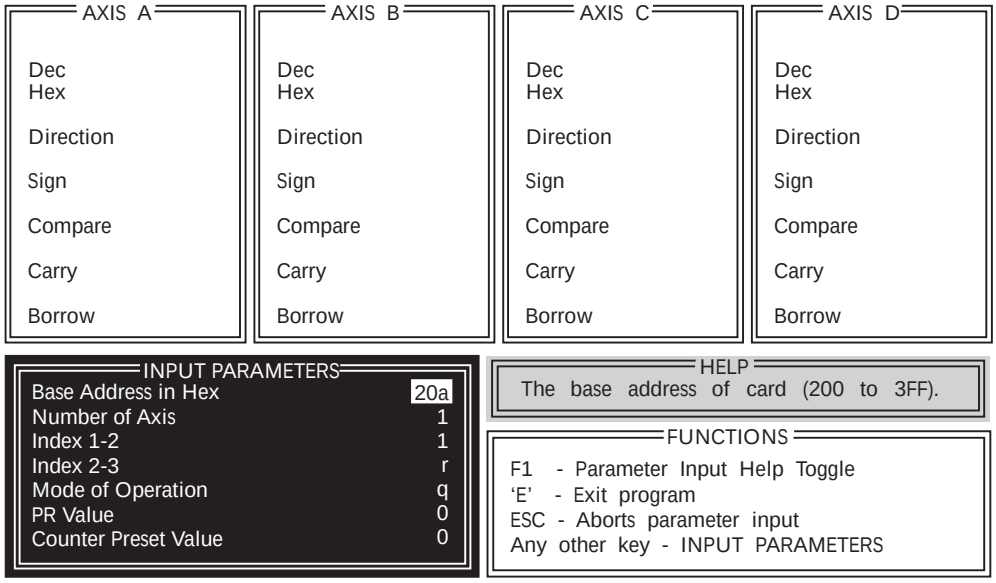
B
Demonstration Program

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 screen shown in Figure B-1 will be displayed:

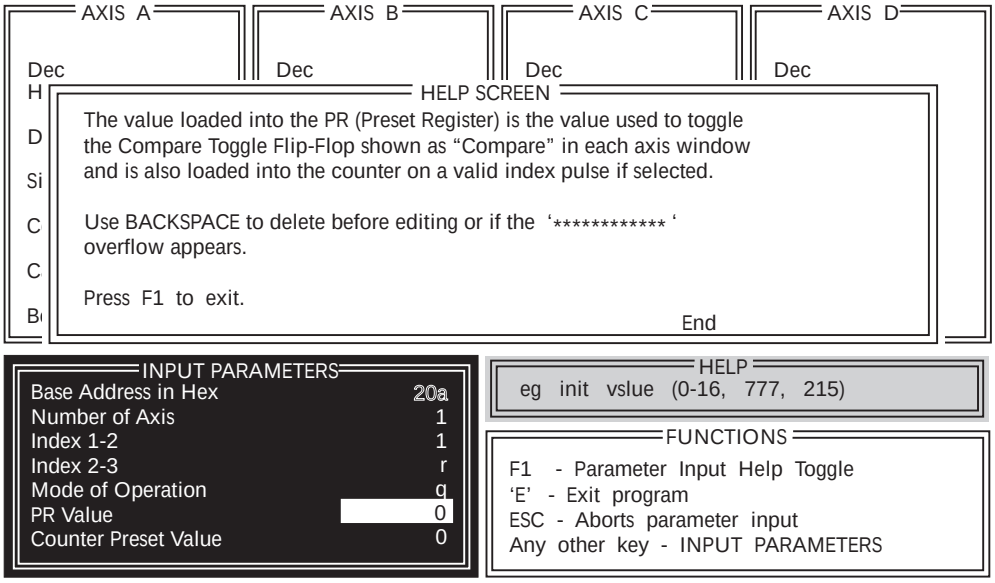
Figure B-1
Input parameters screen



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 <F1>. A help screen will appear containing a definition of the parameter. For example, pressing <F1> when the cursor is at the PR Value will bring up the screen shown in Figure B-2.

Figure B-2
Help 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 overflowed 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.

C

Visual BASIC Demonstration 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.

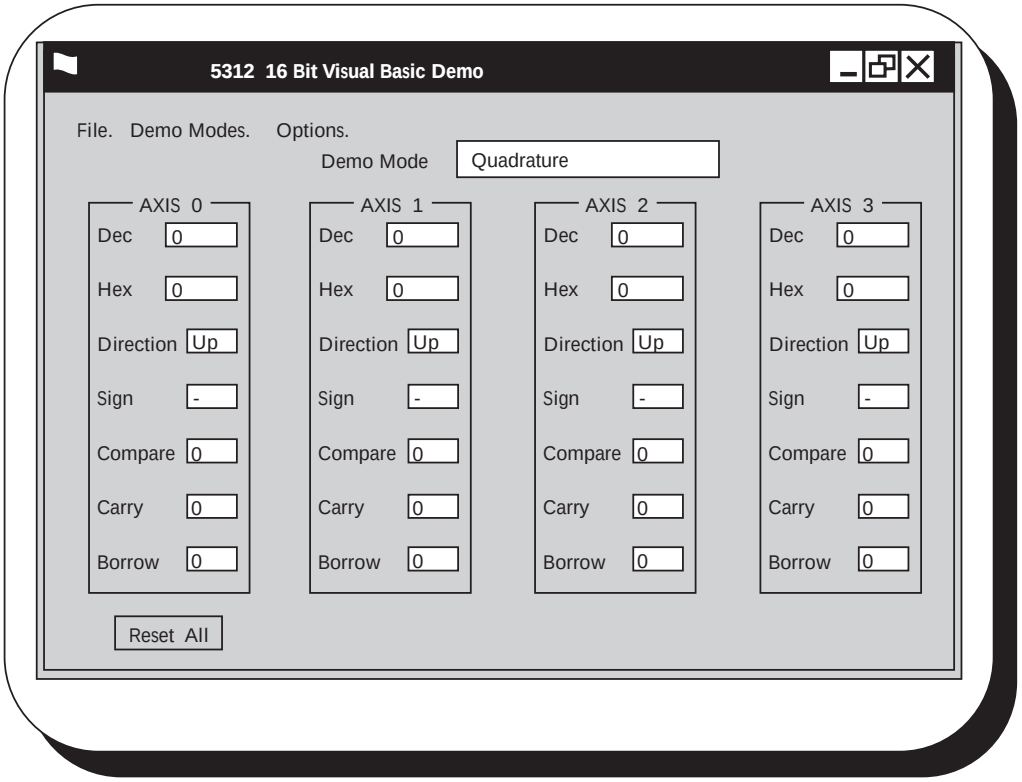
Windows 95

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 C-1.

Figure C-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, use the following procedure.*

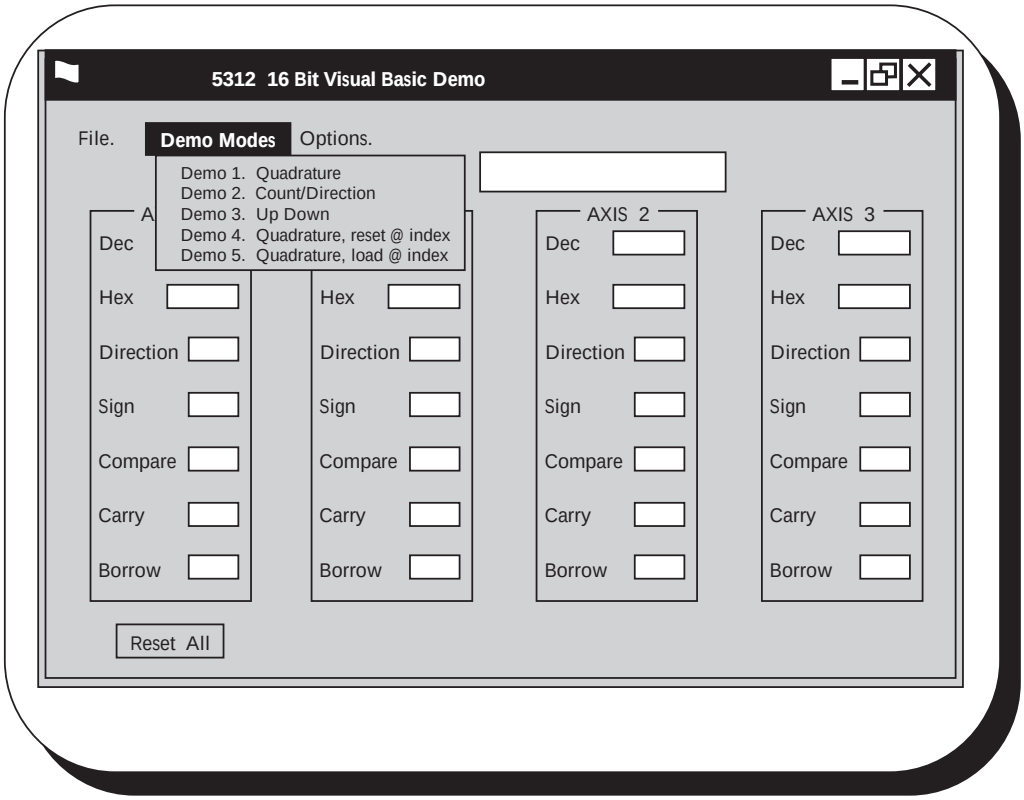
1. *Read the 5312 Technical Reference regarding changing board addresses. In this section, note how the switches on the board enable you to configure the board for a non-default address.*

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

Demo modes

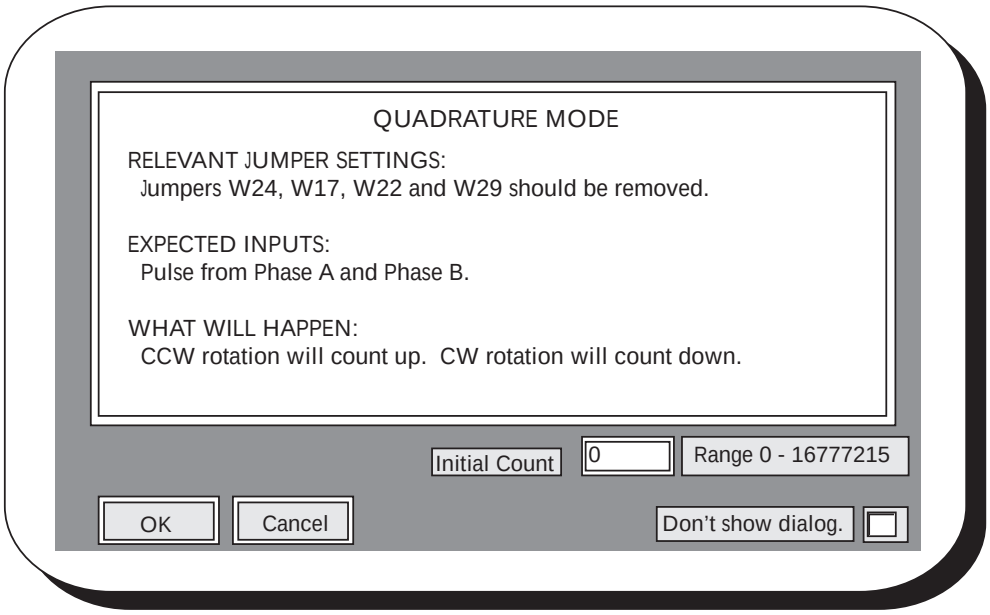
The first logical operation is to select the demo mode you wish to operate in. To do this, pull down Demo Modes as shown in Figure C-2.

Figure C-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 Figure C-3 will appear.

Figure C-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

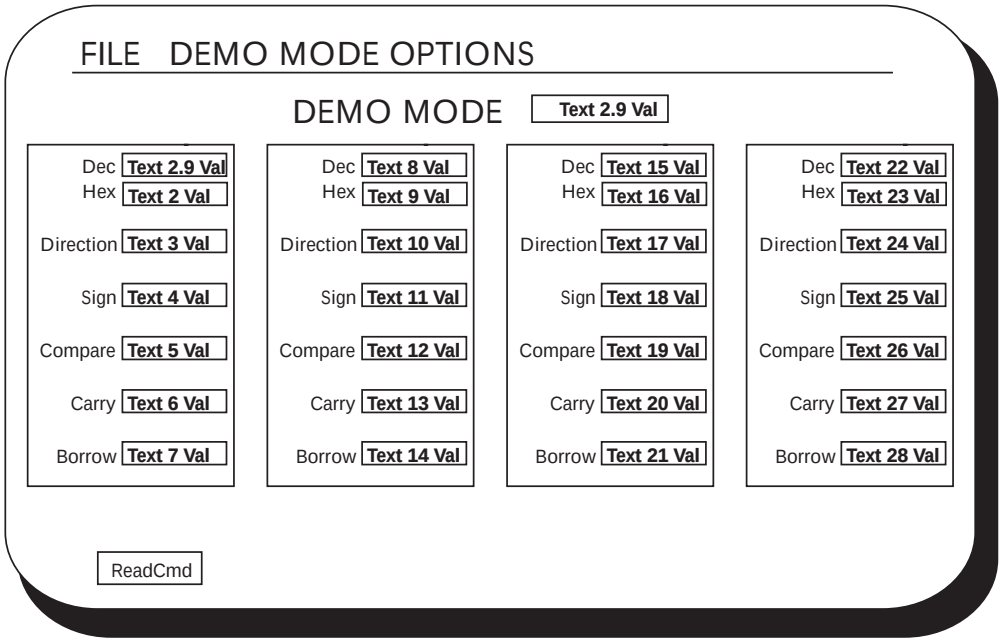
This demo was developed under Visual BASIC 4.0. For those unfamiliar with Visual BASIC, there are two primary types of modules developers create when they write a program: form, and code modules. Form modules specify the form that GUI windows will take. Code modules are the guts of the program. They 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 of 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

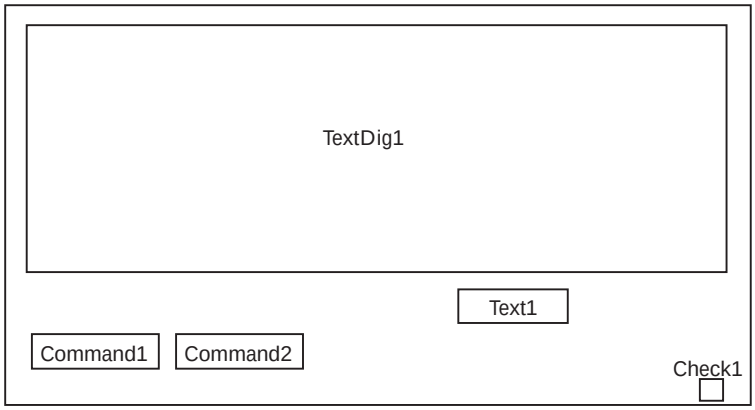
The main form module is FORM12.FRM. It is comprised of menu items, text boxes, and a command button. See Figure C-4.

Figure C-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 C-5 for a template.

Figure C-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.

D

LSI Chip Applications Note

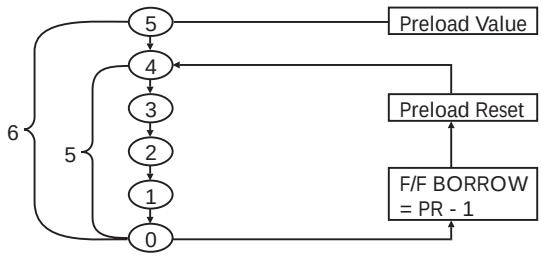
Introduction

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 a +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).

Figure D-1
LSI chipset counter problem



NOTE: This makes the problem associated with low counts per index manifest.

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 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.

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

Index

R

S

T

U

V

W

KEITHLEY

Keithley Instruments, Inc.
28775 Aurora Road
Cleveland, Ohio 44139

Printed in the U.S.A.