KEITHLEY

Application Note Series

Number 2361

Writing Prober Drivers for the Model 4200-SCS Semiconductor Characterization System

This application note provides all the information needed to write a prober driver for Keithley's Model 4200-SCS Semiconductor Characterization System. However, it assumes that the writer of a prober driver is already familiar with programming in C.

Writing a prober driver differs from writing drivers for other external instruments in several ways:

1. Even though there are a variety of prober models, as well as both GPIB and RS-232 communication modes, Keithley provides just four general user functions in the *prbgen* library: *prbInit*, *prbChuck*, *prbMovNxt*, and *prbSSMovNxt*. The *prbInit* function is used to initialize the prober parameters, which will be discussed later. The *prbChuck* function is used to move the chuck to contact or separate position. The *prbMovNxt* function is used to move the prober to the next die, based on a given wafer map file, and the *prbSSMovNxt* function is used to move the prober to the next subsite.
2. The details for a specific prober model are implemented in three places.
   a) A file named *c:\S4200\sys\kcon\sysconfiginfo.ini* contains prober name and prober type, etc. When KCON (Keithley CONfiguration utility) is loaded, this file is used to recognize the prober.
   b) A file named *c:\S4200\sys\dat\prbcnfg_xxxx.dat*, where *xxxx* is the specific prober type in the file described in 2a). This file contains more information about the prober, including communication style, GPIB address, and so on. When KITE (Keithley Interactive Test Environment) is loaded, this file is used to recognize the prober. The file also includes a number of instrument commands, such as GPIB timeout, etc.
   c) A prober-specific library, which includes all the functions necessary to make the prober work properly.

   Note: All prober driver programs should be developed in KULT (Keithley User Library Tool).

The *Prb_t* Structure

A structure named *prb_t*, which is defined in *prb.h*, is used to contain all the parameters of a prober. Refer to **Appendix A** for a listing of this structure. Some of the fields are based on the information provided in the *config_xxxx.dat* file and are taken care of by other routines in KITE. When writing a driver, there's no need to worry about changing the values of such fields in the program. These parameters can be assigned directly, without the need to claim them first. Other fields are for positions and units. The values of these fields may be changed as desired in programs. The most commonly used fields are:

```
int chuck_position; /*Current Chuck Position: init */
int x_position; /*present x location, this is the row number: init, mv */
int y_position; /* present y location, this is the column number: init, mv */
int timeout; /* I/O timeout value*/
int short_timeout; /* I/O timeout value */
int x_initial_location; /* initial x coordinate: init */
int y_initial_location; /* initial y coordinate: init */
int psxl; /* present x subsite location: init */
int psyl; /* present y subsite location: init */
```

For detailed information on the *prb_t* structure, refer to **Appendix A:** Definition of *prb_t* Structure.

KCON

Adding a new prober option in KCON requires modifying a file named *sysconfiginfo.ini*. This file lists all external instruments Keithley supports on the Model 4200-SCS, including C-V meters, switch matrices, and probers. This file is in the *c:\s4200\sys\kcon* directory. Open this file in a text editor like Notepad and scroll down to the prober section, which looks like this:

```
[PROBERS]
DEFAULT=P2MODEL
P1MODEL=Fake Prober
P1DRIVER=FAKE
P1MINPINS=2
P1MAXPINS=360
P1DEFAULTPINS=12
P2MODEL=Manual Prober
P2DRIVER=MANL
P2MINPINS=2
P2MAXPINS=360
P2DEFAULTPINS=12
P3MODEL=Micromanipulator 8860 Prober
P3DRIVER= MM40
P3MINPINS=2
P3MAXPINS=360
P3DEFAULTPINS=12
P4MODEL=Karl Suss PA200 Prober
P4DRIVER=PA200
P4MINPINS=2
P4MAXPINS=360
P4DEFAULTPINS=12
```

Next, add new lines of code to accommodate the new prober. The following example of added code is for a Cascade prober.

```
P5MODEL=Cascade Summit12000 Prober
P5DRIVER=CC12K
P5MINPINS=2
P5MAXPINS=360
P5DEFAULTPINS=12
```

Save the changes and exit KCON. Then, when KCON is re-opened, the user will have the option of adding a new prober that is not previously supported.

Warning: When attempting to reinstall or upgrade the KTEI software, be aware the *sysconfiginfo.ini* file will be refreshed to the default. Be sure to back up the modified *sysconfiginfo.ini* file before reinstalling or upgrading the software. After reinstalling or upgrading, replace the default file with the back-up file.

KITE

When KITE is loaded and run, it refers to KCON for the prober type that is connected to the Model 4200-SCS. Then based on that prober type, KITE loads the *prbcnfg_xxxx.dat* file mentioned previously and copies it to a file named *prbcnfg.dat*. Based on the information provided in that file, KITE will be able to fill in some fields in the *prb_t* structure, such as communication type. Also, when an ITM (Interactive Test Module) is running, KITE will load fields of *x_position* and *y_position* in the *prb_t* structure to update the current column and row position of the wafer. This column and row position is displayed on the *Data->Settings* page of the Model 4200's Graphical User Interface. See the highlighted area of *Figure 1* for an illustration of these site coordinates.

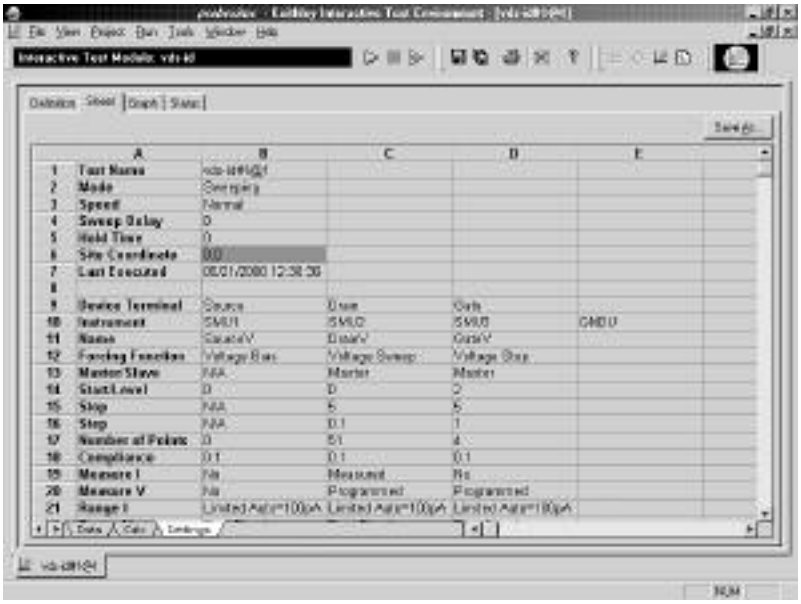

Figure 1.

Prober specific drivers

Five essential programs must be written to run the prober project. These programs are *Cnfg_xxxx.c*, *prbInit_xxxx.c*, *prbChuck_xxxx.c*, *prbMovNxt_xxxx.c*, and *prbSSMovNxt_xxxx.c*, where *xxxx* is the prober type. Before explaining each routine in detail, it's important to become familiar with the subroutine used for communication. Keithley provides a special subroutine named *pruniversalio()* for prober-to-PC communication. This subroutine can be used for either GPIB or RS-232 communication style. Based on the *prb.io_mode* field, the subroutine will use the corresponding communications style to send and receive data.

Include Files

The following files must be included in the prober driver programs.

```
#ifdef WIN32
#include "ktemalloc.h"
#endif
#include <stdio.h>
#include <lptdef.h>
#include <lptdef_lowercase.h>
#include <math.h>
#include    <string.h>
#include    "prb.h"
#include    "prb_msg.h"
#include    "prb_extern.h"
#include    "prbXXXX_proto.h"
#include    "prb_drvadr.h"
#include    "prb_func_id.h"
```

Cnfg_xxxx.c

The basic purpose of this routine is to load prober-specific routines into memory.

```
Protocol definition:
       int Cnfg_xxxx( int teststation );
Main variable definition:
       #ifdef WIN32
              _declspec (dllimport) prb_t Prb[];
       #endif
Pseudo-code:
       /*configure communication*/
       if ( Prb[ teststation ].io_mode == SERIAL )
       confstat = Cnfg_tty( teststation );*/
       else if ( ( Prb[ teststation ].io_mode == GPIB ) ||
              ( Prb[ teststation ].io_mode == GPIBCT ) )
       confstat = Cnfg_gpib( teststation );

       Other variable check (prober specific);

       /* Put other driver routines into memory */
       (void)putdrvadr( teststation, PRINIT, &PrInit_XXXX );
       (void)putdrvadr( teststation, PRCHUCK, &PrChuck_XXXX );
       (void)putdrvadr( teststation, PRMOVNXT, &PrMovNxt_XXXX );
       (void)putdrvadr( teststation, PRSSMOVNXT, &PrSSMovNxt_XXXX );
```

Note:

- This routine is automatically called in KITE. This is not a user visible routine.
- Source codes for functions *Cnfg_tty()*, *Cnfg_gpib()* and *putdrvadr()* are in the *c:\S4200\sys\Prb\Prbcom* directory.
- The function ids in *putdrvadr()* calling parameters, such as *PRINIT*, *PRCHUCK*, are defined in the *c:\S4200\sys\include\prb_func_id.h* file.

prbInit_xxxx.c

This routine is used to initialize prober parameters defined in the structure *prb_t*.

Note: This routine must be called prior to other routines in KITE.

Protocol definition:

```
int PrInit_xxxx( int mode, double x_die_size, double y_die_size, int
       x_start_position, int y_start_position, int units, int subprobtype );
```

Main variable definition:

```
#ifdef WIN32
       _declspec (dllimport) prb_t Prb[];
#endif
int teststation;
```

Pseudo-code:

```
/* Get index for current prober teststation */
teststation = gettstn();

/*initialize parameters*/
Prb[teststation].chuck_position = PR_CHUCK_DOWN;
…

/* Get current column row position */
(void)sprintf(ibuf,"some command string to get current position");
sendstatus = pruniversalio(ibuf, strlen(ibuf), obuf, 60,
       XXXX_TERMINATOR, NUM_XXXX_TERMINATORS, &i_srq,
       Prb[teststation].short_timeout);

Scan the output buffer for column and row position;

/* Initialize the current column row position parameters */
Prb[teststation].x_position = row - x_start_position;
Prb[teststation].y_position = col - y_start_position;
```

prbChuck_xxxx.c

This routine is used to control the chuck position.

Protocol definition:

```
int PrChuck_xxxx( int chuck_position );
```

Main variable definition:

```
#ifdef WIN32
       _declspec (dllimport) prb_t Prb[];
#endif
int teststation;
```

Pseudo-code:

```
/* Get index for current prober teststation */
teststation = gettstn();

/*initialize parameter*/
Prb[teststation].chuck_position = chuck_position;

/*Send command to move the chuck */
if ( chuck_position == PR_CHUCK_DOWN)
       (void)sprintf(ibuf,"some commands to move chuck down");
else if (chuck_position == PR_CHUCK_UP)
        (void)sprintf(ibuf,"some commands to move chuck up");
sendstatus = pruniversalio(ibuf, strlen(ibuf), obuf, 60,
       XXXX_TERMINATOR, NUM_XXXX_TERMINATORS, &i_srq, Prb[teststation].timeout);
```

prbMovNxt_xxxx.c

This routine is used to move the wafer to the next die

Protocol definition:

```
int PrChuck_xxxx( int chuck_position );
```

Main variable definition:

```
#ifdef WIN32
	_declspec (dllimport) prb_t Prb[];
#endif
int teststation;
```

Pseudo-code:

```
/* Get index for current prober teststation */
teststation = gettstn();

/* Send command to move the wafer to next die */
(void)sprintf(ibuf,"some command to move prober to next die");
sendstatus = pruniversalio(ibuf, strlen(ibuf), obuf, 60,
	xxxx_TERMINATOR, NUM_xxxx_TERMINATORS, &i_srq, Prb[teststation].timeout);
/* Get the current prober position */
if ( sendstatus > 0 )
{
	(void)sprintf(ibuf,"some command toget current position ");
	sendstatus=   pruniversalio(ibuf, strlen(ibuf), obuf, 60,
		XXXX_TERMINATOR, NUM_XXXX_TERMINATORS, &i_srq, Prb[teststation].short_timeout);
	if ( sendstatus > 0 )
	{
		scan the output buffer for column and row position;

		/*Update the column and row position */
		Prb[teststation].x_position = i_row –
				Prb[teststation].x_initial_location;
		Prb[teststation].y_position = i_col –
				Prb[teststation].y_initial_location;
	}
}
```

prbSSMovNxt_xxxx.c

This routine is used to move the wafer to the next site. If there are several subsites on the current die, it moves the prober to the next subsite. If the prober is on the last subsite of the current die, it moves the prober to the first subsite of the next die.

Protocol definition:

```
int PrChuck_xxxx( int chuck_position );
```

Main variable definition:

```
#ifdef WIN32
	_declspec (dllimport) prb_t Prb[];
#endif
int teststation;
```

Pseudo-code:

```
/* Get index for current prober teststation */
teststation = gettstn();

/* Send command to move the wafer to next site*/
(void)sprintf(ibuf,"some command to move prober to next site");
sendstatus = pruniversalio(ibuf, strlen(ibuf), obuf, 60,
	xxxx_TERMINATOR, NUM_xxxx_TERMINATORS, &i_srq, Prb[teststation].timeout);

/* Get the current prober position */
if ( sendstatus > 0 )
```

```
{
        (void)sprintf(ibuf,"some command to get current position ");
        sendstatus=          pruniversalio(ibuf, strlen(ibuf), obuf, 60,
                XXXX_TERMINATOR, NUM_XXXX_TERMINATORS, &i_srq, Prb[teststation].short_timeout);
        if ( sendstatus > 0 )
        {
                scan the output buffer for column and row position;

                /*Update the column and row position */
                Prb[teststation].x_position = i_row –
                                Prb[teststation].x_initial_location;
                Prb[teststation].y_position = i_col –
                                Prb[teststation].y_initial_location;
        }
}
```

Note: Depending different models of probers, this approach may be modified as needed.

Appendix A: Definition of *prb_t* Structure

```
typedef struct
{
        int chuck_position; /*Current Chuck Position: init */
        int x_position; /*present x location: init, mv, setrefdie */
        int y_position; /* present y location: init, mv, setrefdie */
        int psxl; /* present x subsite location: init */
        int psyl; /* present y subsite location: init */
        int mode; /* present probing mode: init, setmode */
        int probtype; /* prober type */
        char probname[PRB_LIBNAME_LEN]; /* prober's name (replaces type) */
        int subprobtype; /* prober sub-type: init */
        int ready; /* ready to probe?: init */
        int movestatus; /* Move Successful?: init */
        int x_index; /* number of unit size steps in x index: init */
        int y_index; /* number of unit size steps in y index: init */
        int x_initial_location; /* initial x coordinate: init, setrefdie */
        int y_initial_location; /* initial y coordinate: init, setrefdie */
        int x_move_step; /* number of ticks in x step: init */
        int y_move_step; /* number of ticks in y step: init */
        int units; /* prober units: init, setunits */
        int number; /* prober number (spelled funny) */
        int error_level; /* current error reporting level */
        char error_log_name[MAXFILENAMESIZE]; /* error log file name */
        FILE *error_log_fp; /* file pointer for error log file */
        FILE *trans_log_fp; /* file pointer for transaction log file */
        FILE *fake_log_fp; /* file pointer for FAKE debug only */
        char trans_log_name[MAXFILENAMESIZE];/* transaction log file name */
        char fake_log_name[MAXFILENAMESIZE]; /* fake log file name */
        int trans_log_enabled; /* Transaction Log Flag */
        int fake_log_enabled; /* fake Log Flag */
        int file_des; /* file descriptor */
        int io_mode; /* ex: SERIAL GPIB */
        char RS232_device[TTY_DEV_NAME_LEN];
        char baudrate[TTY_DEV_NAME_LEN];
        int timeout; /* I/O timeout value, settime */
        int device_irq; /* i/o device interrupt */
        int gpib_unit;          /* converter box unix number */
        int gpib_slot;          /* gpib slot number */
        int gpib_address;     /* gpib address */
        int gpib_writemode;   /* gpib writemode */
        int gpib_readmode;    /* gpib readmode */
        int gpib_terminator;  /* gpib terminator */
        char s_prb_options[MAX_OPTIONS_SIZE]; /* prober options:check option*/
        double d_cur_pos_x;   /* current x subsite accum. rel move: init, relmv, relret*/
        double d_cur_pos_y;   /* current y subsite accum. rel move: init, relmv,relret */
        double d_abs_site_x;  /* current y site location abs move: init, absmv */
        double d_abs_site_y;  /* current y site location abs move: init, absmv */
        int i_prb_max_slots;  /* max slots in a cassette from prbcnfg*/
```

```
        int i_prb_max_cassettes; /* max cassettes on a prober from prbcnfg*/
        int short_timeout; /* I/O timeout value */
        int smif_lock_status; /* status of the lock on the pod: init */
        int smif_clamp_status; /* status of the clamp on the pod: init */
        int smif_cassette_status; /* status of the cassette on the pod: init, prbstat */
        int smif_machine_status; /* status of the machine: init, prbstat */
        int smif_sense_wafer; /* sense wafer(s) in cassette: init, sense wafer */
        int smif_stop_resume; /* probing stop(ped)/resume(d): init */
        double d_die_size_x;  /* x die size real # of mm/mils: init, setdiesz */
        double d_die_size_y;  /* y die size real # of mm/mils: init,setdiesz */
        double d_init_coord_x;  /* initial X machine coord: init */
        double d_init_coord_y;  /* initial Y machine coord: init */
} prb_t ;
```

Appendix B: A Sample *prbcnfg_xxxx.dat* File

```
# prbcnfg_PA200.dat - DEFAULT Prober Configuration File
#
# The following tag, "PRBCNFG", is used by the engine in order to determine
# the MAX number of SLOTS and CASSETTES for a given prober at runtime.
<PRBCNFG>
# for OPTIONS "" == NULL, max 32 chars in string
# Example
#              01234567890
#PROBER_1_OPTIONS=1,1,1,1,1,1
#      OcrPresent
#      AutoAlnPresent
#      ProfilerPresent
#      HotchuckPresent
#      HandlerPresent
#      Probe2PadPresent
# Configuration for PA200 probers:
# PA200
#PROBER_1_PROBTYPE=PA200
#PROBER_1_OPTIONS=0,0,0,0,1,0
#PROBER_1_IO_MODE=SERIAL
#PROBER_1_DEVICE_NAME=COM1
#PROBER_1_BAUDRATE=9600
#PROBER_1_TIMEOUT=300
#PROBER_1_SHORT_TIMEOUT=5
#PROBER_1_MAX_SLOT=25
#PROBER_1_MAX_CASSETTE=1
# Configuration for direct GPIB probers:
# PA200
#
PROBER_1_PROBTYPE=XXXX
PROBER_1_OPTIONS=0,0,0,0,1,0
PROBER_1_IO_MODE=GPIB
PROBER_1_GPIB_UNIT=0
PROBER_1_GPIB_SLOT=1
PROBER_1_GPIB_ADDRESS=28
PROBER_1_GPIB_WRITEMODE=0
PROBER_1_GPIB_READMODE=2
PROBER_1_GPIB_TERMINATOR=13
PROBER_1_TIMEOUT=300
PROBER_1_SHORT_TIMEOUT=5
PROBER_1_MAX_SLOT=25
PROBER_1_MAX_CASSETTE=1
#
```

KEITHLEY

A GREATER MEASURE OF CONFIDENCE

KEITHLEY INSTRUMENTS, INC. ■ 28775 AURORA ROAD ■ CLEVELAND, OHIO 44139-1891 ■ 440-248-0400 ■ Fax: 440-248-6168 ■ 1-888-KEITHLEY ■ www.keithley.com

BELGIUM
Sint-Pieters-Leeuw
Ph: 02-363 00 40
Fax: 02-363 00 64
www.keithley.nl

CHINA
Beijing
Ph: 8610-82255010
Fax: 8610-82255018
www.keithley.com.cn

FINLAND
Espoo
Ph: 09-88171661
Fax: 09-88171662
www.keithley.com

FRANCE
Saint-Aubin
Ph: 01-64 53 20 20
Fax: 01-60 11 77 26
www.keithley.fr

GERMANY
Germering
Ph: 089-84 93 07-40
Fax: 089-84 93 07-34
www.keithley.de

INDIA
Bangalore
Ph: 080-22 12 80-27/28/29
Fax: 080-22 12 80 05
www.keithley.com

ITALY
Milano
Ph: 02-553842.1
Fax: 02-55384228
www.keithley.it

JAPAN
Tokyo
Ph: 81-3-5733-7555
Fax: 81-3-5733-7556
www.keithley.jp

KOREA
Seoul
Ph: 82-2-574-7778
Fax: 82-2-574-7838
www.keithley.co.kr

MALAYSIA
Kuala Lumpur
Ph: 60-3-4041-0899
Fax: 60-3-4042-0899
www.keithley.com

NETHERLANDS
Gorinchem
Ph: 0183-63 53 33
Fax: 0183-63 08 21
www.keithley.nl

SINGAPORE
Singapore
Ph: 65-6747-9077
Fax: 65-6747-2991
www.keithley.com.sg

SWEDEN
Solna
Ph: 08-50 90 46 00
Fax: 08-655 26 10
www.keithley.com

SWITZERLAND
Zürich
Ph: 044-821 94 44
Fax: 41-44-820 30 81
www.keithley.ch

TAIWAN
Hsinchu
Ph: 886-3-572-9077
Fax: 886-3-572-9031
www.keithley.com.tw

UNITED KINGDOM
Theale
Ph: 0118-929 75 00
Fax: 0118-929 75 19
www.keithley.co.uk

Printed in the U.S.A. No. 2361 01073KGW