WAVECREST Corporation

PROGRAMMING THE WAVECREST DTS 2070 IN "C" AND "PASCAL"

Application Note No. 113

WAVECREST Corporation continually engages in research related to product improvement. New material, production methods, and design refinements are introduced into existing products without notice as a routine expression of that philosophy. For this reason, any current *WAVECREST* product may differ in some respect from its published description but will always equal or exceed the original design specifications unless otherwise stated.

***WAVECREST* Corporation**
A Technologies Company
7275 Bush Lake Road
Edina, Minnesota 55439
(612) 831-0030
(800) 733-7128
www.wavecrestcorp.com

WAVE TECHNOLOGIES APPLICATIONS NOTE NUMBER 113

PROGRAMMING THE WAVE DTS2010 IN "C" and "PASCAL"
June 15, 1993

Introduction

The WAVE Digital Time Scope DTS2010 is capable of making very fast, accurate time measurements and can be an excellent instrument for the automated measurement environment used for Integrated circuit(IC) testing.

Most automated applications using the WAVE DTS involve programming software to drive the instrument on the GPIB BUS. The GPIB commands are similar for most instruments, but the main difficulty lies in programming languages. Whether the computer is an IBM-486 based PC, SUN/SPARC, or HP/APOLLO workstations; the programming language is usually never the same.

Each computer will have its own primative software drivers that drive the GPIB bus and most of the drivers are either custom or generic (like National Instruments NI-488) with different software languages.

Furthermore to complicate the software language issue; many of the instruments IEEE488 user manuals will have software examples in some other language. Thus the customer is expected to do translations while writing code.

This applications note describes some of the GPIB code necessary in programming the DTS in "C" or "PASCAL" and compares different data types, formats, and the ASCII serial command streams for **talk** and **listen**.

GPIB/IEEE-488.2 standard

The GPIB/IEEE-488.2 standard has a common set of commands that define protocol, status reporting, error handling, and data formats. The DTS2010 instrument conforms to the standards of IEEE-488.1 and IEEE-488.2 and will interface to computers and controllers that comply to these standards.

GPIB program

Every GPIB program has code to initialize and setup the instrument, serial poll the BUS, and execute the measurements. The code will have "C" functions or "PASCAL" procedures that are declared in *.h or *.ins.pas, *.ins.test INCLUDE files. The data stream sent to the DTS must be converted into strings and the strings received from the DTS must be converted to data of the proper type, whether it be integer, real, or character strings.

Once the data is converted, it will be used in some calculations, loops, and decisions made based on the results. In this application, the various elements of the software language will be explored.

Instrument Commands

All serial streams are converted to ASCII strings in "C" or "PASCAL" and sent to the instrument via the GPIB BUS. For most applications, the recommended command sequence for initialization of the DTS and setup parameters to execute a time measurement are as follows :

```
  INITIALIZE DTS2010 COMMANDS
  *CLS                      ( Clear the Status command clears the status registers )
   :SYST:HEAD OFF           ( No header or code characters to Controller )
   :SYST:LONG OFF           ( Abbreviated mnemonic is the first four characters )
  *ESR?                     ( Query to read the event status register  )

'PRAMETERS SETUP OF DTS2010 COMMANDS
   :TRIG:SOURAUTSTOP       (  Set trigger source to automatic and auto arm on stop channel )
   :ACQ:COUN000100         (  Set sample size to 100 counts )
   :SYST:CHANBOTH          (  Set channel to measure both or start channel to stop channel )
   :ACQ:FUNCTPD++          (  Set function to TPD++ or both channels edges rising to rising )
   :SYST:TERMSTAR-2.0000 (  Set start channel termination to -2.000vdc ECL )
   :SYST:TERMSTOP0.0000  ( Set stop channel termination to  0.000vdc TTL  )
   :CHANSTAR:LEV-1.0000   (  Set start channel 1 edge level to -1.000vdc )
   :CHANSTOP:LEV1.0000    (  Set stop channel 2 edge level to +1.000vdc )

   BURST MEASUREMENT OF DTS2010 COMMANDS
   :TER?                  (  Query to read the status event register, 10 = complete )
  *TRG                    (  Trigger command to initiate a measurement )
   :TER?                  (  Query to read the event register, 10 = complete )
   :MEAS:AVER?            (  Read average measurement  )
```

In this application a wide range of GPIB software commands are used and described in this paper.

Declarations of variables

In "C" all voltage parameters, whether termination or edge level voltage are type **real** and declared as **double**. All burst measurements, whether AVERage, SDEV, JITTer, RANGe, MINimum, MAXimum are **floating point** and declared as **double**.

```
double voltage1, voltage2;
double average1, jitter1;
```

In "PASCAL" all voltage parameters are type **real** and declared as **real**. All burst measurements are **exponential** and declared as **double**.

```
average1, jitter1     : double;
voltage1, voltage2  : real;
```

Data type modifiers

In "C" all sample sizes are type **integer** using the type modifier long and declared as **long**.

```
long no_of_samples;
```

In "PASCAL" all samples sizes are type **integer** and declared **integer32.**

```
no_of_samples : integer32;
```

High level talk functions

The structure of programming DTS commands of high level functions that pass GPIB address information and instrument commands to low level functions or primatives is shown below. The high level function call can be the same in "C" or "PASCAL ".

talk_488(adr, " :TRIG:SOURAUTSTOP");

All programming commands will use the function **talk_488** to send commands to the BUS and receive a status report response that the command is complete.

Low level talk functions

The low level functions or primatives are routines that do all the coding and monitoring of status report responses. They convert the DTS commands to strings, perform queries on the BUS, and timeout if interrupted or the commands are not properly terminated. The low level functions are transparent to the high level function and some of the GPIB drivers provide these low level functions as part of the software.

An example of a "C" low level function **talk_488** to the NI-488 driver is shown below:

```
int talk_488(adr,command)
int *adr;
char *command;
{
ibwrt(adr, command, (long) strlen( command ));
if (ibsta < 0) return(-1);
else
  return(0);
}
```

As you can see, low level functions are cryptic and depending on the specific GPIB driver; some functions will actually write to board numbers and registers. Basically in the example, **ibwrt** writes the address and string using the function **strlen** to the BUS**.** Strlen is a special function that returns the length in characters of a string called **command**.

In "PASCAL" converting the command to a string involves putting the command into a data structure type of an array called **data**.

```
procedure talk_488 (IN adr : integer; IN command : string_130;)
var
strg_len : integer;
 adr1, x, y :  integer;
  command, data  : string_130;
 begin
   adr1 :=  adr;
   strg_len := 255;

while command[strg_len] = ' ' do strg_len := strg_len - 1;
   x := strg_len;
   strg_len := 1;
   y := 1;
while strg_len <> (x + 2) do
begin
   if (command[strg_len} <> ',') and (strg_len <> (x+1) then
   begin
      data[y] := command[strg_len];
      y := y + 1;;
  end
```

```
    else
     begin
                gpib_wr (adr1, data, y-1);
                  serial_poll();
                  error_report();
                timeout();
      end;
     strg_len := strg_len + 1;
    end;
end;    { talk_488 }
```

In the "PASCAL" procedure example, **gpib_wr** writes the address **adv1** and command string indexed variable array **data** to the DTS, waits for responses that the operation is complete **serial_poll**, and monitors the **timer**.

High level listen functions

The high level function for **listen_488** is similar to **talk_488**. All instrument commands and queries are sent to the BUS in order to receive a response message from the DTS. The syntax of a query is identical to a command except the query is always followed by a question mark(?) character.

The example below issues a **talk_488** command for measurement (*TRG) and the function **listen_488** for the response measurement **average** data(:MEAS:AVER?) returned from the DTS. The high level function call can be the same in "C" or "PASCAL ".

```
talk_488( adr, " :TER?, *TRG, :TER?");
listen_488(adr," :MEAS:AVER?", &average1);
listen_488(adr," :MEAS:JITT?", &jitter1");
```

The **jitter** measurements(:MEAS:JITT?) are essential to show integrity of the signal path and should be included with every parameter. A high jitter measurement parameter can indicate a problem with contact resistance or a noisy ground connection.

Low level listen functions

The low level functions or primitives are routines that do all the coding of commands, monitoring of status, and obtain the responses or measurement data from the DTS.

An example of a "C" device trigger and response low level function **listen_488** to the NI-488 driver is shown below :

```
int  listen_488(adr,command, double_response)
int *adr;
char *command;
double *double_response;
{
float temp_double;
char *tmp_str;
char scratch[20];

request_and_obtain_asc(adr, command, scratch);
if (( tmp_str = ( char *)strstr ( scratch , "+" )) != null )
{
 sscanf ( tmp_str, "+%f", &temp_double);
}
else
if (( tmp_str = ( char *)strstr ( scratch , "-" )) != null )
{
```

```
 sscanf ( tmp_str, "-%f", &temp_double);
  temp_double = -temp_double;
}
else
   sscanf ( tmp_str, "%f", &temp_double);
 *double_response = temp_double;
return(0);
}
```

Once again, low level functions are cryptic and depend on the specific GPIB driver provide by the vendor. Basically in the example, **request_and_obtain_asc** sends an **ibwrt,** which writes the address and command string to the BUS**.** The function also does a **ibrd**, that reads the DTS response string. The **strstr** is a special function that matches the first occurrence of the string and returns a null pointer if no match was found for the polarity sign. **sscanf** receives a string and converts it to a floating point number and double precision **double_response** is returned.

In "PASCAL" converting the command to a string involves putting the command into an array.

```
procedure listen_488 (IN adr : integer; IN command : string_130;  OUT message : string_30);
var
strg_len, mess_len : integer;
 adr1, x, y :  integer;
  command, message  : string_130;
 begin
   adr1 :=  adr;
   strg_len := 255;
while command[strg_len] = ' ' do strg_len := strg_len - 1;
     talk_488(adr1, command, strg_len);
       gpib_rd(adr1, message, mess_len);
          writeln(' returned message string', message:mess_len);
         serial_poll();
      error_report();
    timeout();
end;
```

average2 := strn_to_real (average1)

In the "PASCAL" procedure example, **gpib_wr** writes the address **adv1** and command string(:MEAS:AVER?) to the DTS, then performs a querie and waits for a response(&average1).

Once the string is received, it must be converted to a **real** using the function **strn_to_real ();** which parses through a string and determines polarity, **exponential** (e) sign, digits and converts to a real number.

Termination characters

All serial stream data transfers must be null-terminated by a GPIB EOI or the NL character. The character for a NL (newline) is a ASCII 0A (hex). and EOI (end or identify) is a hardware line. The parser automatically detects the character and terminates the data transfer.

Serial polling

init_488 function passes serial polling commands that can be performed at setup to accomplish status reporting. This is to assure operations are complete and there are no errors.

The high level function call can be the same in "C" or "PASCAL ".

init_488(adr, " *ESE 125,*SRE 48, *ESR?");

The event status (ESE) and service request enable (SRE) commands set up masking to obtain status information. Included in the function is exchange protocol to query the event status register (ESR?).

In the example below, **opc_poll_queue ();** is a low level "C" function that performs a query and reports on the event status. The polling status monitors the event status until the operation is complete.

```
int opc_poll_queue() {
    int poll_status;
          while (!(poll_status  & 0x 41))
           {
            if (poll_the_dts(&poll_status))
             return(-1);
            time_out();
            }
   return(0);
    }
```

In the "PASCAL" example below, the procedure **serial_poll( );** checks the DTS for operation complete(OPC), and no errors.

```
procedure serial_poll( IN adr : integer; OUT error : boolean);
var
   poll_status : char;
begin
     gpib_poll(adr, poll_status);
 if ((ORD(poll_status)& 16#0041) = 16#0041 then
   wave_ready := true;
    if wave_ready then
       error := true
   else
        error := false;
end;
```

Conclusions

The IEEE 488.2 standard simplifies the programming software necessary to drive the DTS instrument and perform measurements. The structure of programming is broken into high level calls for **talk** and **listen.** With the use of high level functions, the serial stream of commands are easily programmed in any of the languages of "C" or "PASCAL".

In the low level functions resides the routines that do the code for converting the commands to strings, monitoring the report status, and performing queries on response information. Many of the low level functions are provided by the vender and can be part of the GPIB driver.

All of the examples can be used with the WAVE Digital Time Scope DTS2010 to integrate with any powerful computer or workstation for test and measurement of integrated circuits (IC).

References

Microsoft Corp., Microsoft C Compiler for the MS-DOS Operating System Run-time Library reference manual, 1987.

National Instruments Corp., LabWindows User Interface Library Reference Manual, 1991.

Herbert Schildt, ANSI C Made Easy, Osborne Mcgraw-Hill, Berkeley,Ca.; 1990.

Wave Technologies Corp., Applications Note No. 110; Verifing ATE System Accuracy, June,1993.

Operators Guide, Digital Time Scope, Wave Technologies Corporation, Edina, Mn.

IEEE-488 Interface Guide, Digital Time Scope, Wave Technologies Corporation, Edina, Mn.

George W. Cherry, Pascal Programming Structures, Reston Publishing Co., Reston,Va.;1980.

WAVECREST Corporation
World Headquarters
7275 Bush Lake Road
Edina, MN 55439
(612) 831-0030
FAX: (612) 831-4474
Toll Free: 1-800-733-7128
www.wavecrestcorp.com

WAVECREST Corporation
West Coast Office:
1735 Technology Drive, Suite 400
San Jose, CA 95110
(408) 436-9000
FAX: (408) 436-9001
1-800-821-2272

200113-01 REV A