WAVECREST Corporation

PROGRAMMING THE *WAVECREST* DTS IN A UNIX C/C++ ENVIRONMENT

Application Note No. 116

WAVECREST APPLICATIONS NOTE NUMBER 116 rev. 1.1

PROGRAMMING THE WAVECREST DTS-2070 IN A UNIX - C/C++ ENVIRONMENT

Introduction

The need to make timing measurements at 10pS accuracy and frequency measurements of 1.0GHZ or less are very common in high performance IC test applications. In response to customer demands, the WAVECREST Digital Time Scope DTS2070 or DTS is used with many ATE Testers to make fast, accurate timing measurements that ATE systems can not make; because of either a basic accuracy or bandwidth (BW) limitations.

Most ATE Test systems are using UNIX - based workstations like the SUN in a C/C++ environment with the NI-488.2M GPIB software driver with the NI interface. This paper is for the Test Engineer to help write a test program to measure critical timing parameters on I.C.'s; using the DTS with a ATE tester in a automated environment via the GPIB bus. This application note explains how to :

1) Configure the GPIB device parameters in a NI-488 configuration file for the GPIB software driver.
2) Include files that must be declared in order to perform the necessary function calls of the GPIB software driver.
3) Initialize and setup the DTS with C/C++ software commands.
4) New quick easy to use setup and measurement macro commands for the DTS.

There are basic fundamental GPIB software commands a user must perform or the controller/instrument will not communicate. This paper is formulated to provide the essential software examples, so the Test Engineer will be successful in executing program instructions over the GPIB.

ANSI/IEEE-488.2 standard

The ANSI/IEEE Standard 488.2 was released in 1987 with a common set of commands that define protocol, error handling, status reporting, and data formats. All ATE test systems have a GPIB/IEEE-488 bus available for interfacing to instruments that adhere to these standards; whether the interface connects to the **SBus** slots or the **SCSI** port.

The DTS2070 instrument conforms to the standards of IEEE-488.1 and IEEE-488.2 and will **talk/listen** to all ATE systems controllers compliant to these standards.

Configuration file

The configuration file defines essential GPIB device parameters that are characteristic of the instrument. I emphasize the fact that without a configuration file the controller/instrument may not communicate. The configuration parameters include primary GPIB address, time-out settings, EOS byte, and bus timing; which are personalized for that instrument. Therefore the file name "DTS207X" will always be associated to the instrument. The configuration file usually resides at /usr/bin or /dev/gpib.o for the UNIX - based workstations and can be configured by the command **ibconf**. An example of the configuration file for the "DTS207X" is as follows :

```
Board: gpib0
Primary gpib address .....00h
Secondary gpib address .....none
Timeout setting.....T10s
EOS byte.....no
Terminate read on EOS .....no
Set EOI with EOS on write .....no
```

Type of compare on EOS 7-bit
Set EOI w/last byte of write..... yes
Board is system controller..... yes
Disable auto serial polling ..... yes
High-speed timing ..... no
UNIX signal..... 2

Device: DTS207X Access: gpib0
Primary gpib address..... 05H
Secondary gpib address ..... none
Timeout setting T10s
EOS byte 00H
Terminate read on EOS..... no
Set EOI with EOS on write..... no
Type of compare on EOS 7-bit
Set EOI w/last byte of write..... yes

Other parameters :
Serial poll timeout 1s
Assert REN when SC ..... no
Bus timing 500nS
Enable repeat address..... no
Parallel poll ..... default
Enable CIC protocol..... no

The software module takes into account any special configuration requirements for the instrument it controls and is layered between the GPIB board (gpib.0) and NI-488.2M GPIB DRIVER as show in Diagram no. 1.

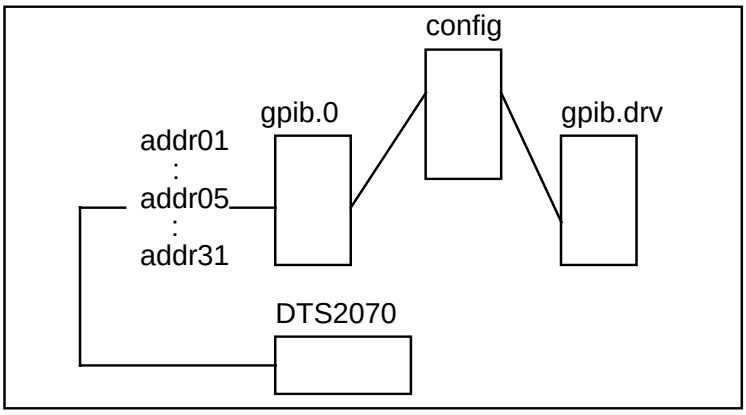

Diagram No. 1

GPIB address

The GPIB has available both primary and secondary addresses. The DTS has 32 selectable gpib addresses (1 - 32). Each instrument has its own unique address and is specified in the configuration file. The example in diagram no. 1 is address 05. Once the DTS address is selected; it is stored in non-voltile memory during power off and remains at that address until it is manually changed.

Include files

At the beginning of the DTS module are necessary include files that need to be declared and are essential to execute the high level function calls of the gpib.drv. It is important to use the proper syntax for the function calls such as ibrd or ibwrt; but without the include file the call would be meaningless and the compiler would detect an error warning. The include file is a preprocessor directive, so the compiler can perform some preparatory work on the source code before compiling,

The include files are as follows :

```
#include  <stdio.h>
#include <strings.h>
#include  "cib.h"          /* gpib ibxx function library for c */
#include  "dts2070.h"    /* special calls supplied by WAVE */
```

Others include files depending on system software :

```
#include <teradyne.h>  /* custom ni driver j953 type testers */
#include  "globals.h"
#include  <sgtty.h>
#include  "ni_gpib.h"     /* custom ni driver for vista, duo, lt1101*/
#include <stdlib.h>
#include <mtbox.h>
#include  "ugpib.h"       /* custom ni driver  se212 type testers */
#include "agile.h"
#include  "ugpib.h"       /* custom ni driver s9000 type testers */
#include  "gpib.h"        /*  ni driver in C for sun  OS, Solaris*/
static int dts_no;
static char rdbck[256];
```

Initialization Function

The DTS needs to be initialized only once at the beginning of the program. Without the initialization and configuration the controller/instrument will not communicate. The common commands at initialization are sent to the DTS by high level function calls of the NI488.2 software driver (gpib.drv). The calls of ibrsp(serial poll response), ibsta(command status), ibrd (read), ibwrt(write), ibtmo(time-out), ibpad(primary address), ibsad(secondary address), ibeos(end of string), ibeot(eoi on last byte), ibclr(clear bus) are all part of this driver(gpib.drv). The **dts_initialize** function is essential to : call the config file "dts207X", communicate to GPIB address 05, bring the instrument on line **ibonl**, clear the GPIB bus **ibclr** and status registers, and allow a 3.0 Sec. delay for the clear to take affect. This dts_initialize function works for all testers, so use the code structure as follows :

```
void dts_initialize( ) {
  if ((dts_no = ibfind ("dts207X")) < 0)
  {
    printf (" GPIB initialization FAILED for BAD address or \n");
    printf (" GPIB cable connection or \n");
  ibonl (dts_no, 0);                          /*  takes instrument off line = 0 */
  }
 else {
  ioctl (dts_no, IBGET, &device);     /*   optional statement associated with ibfind device*/
  ibtmo (dts_no, T10s);               /*   time out 10 sec.          */
  ibpad (dts_no, 5);                  /*   primary address = 5       */
  ibsad (dts_no, 0);                  /*   secondary address = 0  */
  ibeos (dts_no, 0);                  /*   ignore eos char on read = 0 */
  ibeot (dts_no, 1);                  /*   send eoi on last byte of write = 1 */
  ibonl (dts_no, 1);                  /*   instrument on line = 1 */
```

```
if ( ibclr (dts_no) & ERR)
        printf ( " ERROR in BUS CLEAR \n");    /* clear gpib bus  */
  delay ( 3.0);                          /*   delay 3 sec. to allow both bus to clear  */
  if (ibwrt (dts_no, "*CLS",4)) & ERR)
  {
  printf ("\n***** GPIB CABLE or WAVE DTS ADDRESS ERROR ***** \n" );
  return FALSE;
  }
  ibwrt (dts_no, "*CLS; :SYST:HEADOFF; :SYST:LONGOFF; *ESE125; *SRE49",51);
  ibwrt (dts_no, ":ACQ:COUN010; :SYST:WAVPEAK",27);
 }
 return;
 }
```

ibwrt passes the arguments of the GPIB device address with the ASCII command string and **sizeof** the string. Each ASCII command will perform commands that clear the status register (*CLS), header information is **off** (:SYST:HEAD OFF), and the abbreviated truncation rule of 4 characters for alpha mnemonics(:SYST:LONG OFF) is used throughout this application. The event status (*ESE125) and service request enable (*SRE49) commands set up masking to obtain status information during serial polling. The instrument commands (:ACQ:COUN100) set the sample size to 100 and (:SYST:WAVPEAK) sets the pulse find wave for voh/vol measurements. The first ***CLS** exchanges protocol to query status of command transfer to the DTS instrument and report if any **ERR** occurred (**ERR**or is a NI-488 gpib.drv declared variable and has built in error reporting).

Once the function is complete to initialize, invoke header format, setup masking for status reporting, acquire count, and report error status ; the next task is read a response.

Identify Function

First we must read a response (ibrd) from the GPIB. A ***IDN?** command will assure the GPIB communication of the DTS. **ID**e**N**tify query will return a string " WAVE, DTS2070, 0x.x"; While waiting for the reply, I suggest performing a serial poll on the status byte register to check that the **MAV** bit is set high, when ready perform the **ibrd**. The **M**essage **AV**ailable status bit indicates that the GPIB has data to transfer and can be read only after the bit is high. Without the serial poll the data may not be available and the read would be meaningless.

```
void dts_identify( ) {
  ibwrt (dts_no, "*IDN?",5);
  if (ibsta & ERR) printf (" IBSTA ERROR \n" );
        int poll_status;
        poll_status = 0;
                while (!(poll_status  & 0x 10))
                 {
                  ibrsp(dts_no,&poll_status);
                 }
        ibrd (dts_no, rdbck,40);
        printf (wave idn = %s \n",rdbck);
 return;
 }
```

Once the proper response is returned from the DTS; the next step is to use :**SYST**EM: **MAC**RO commands to setup the DTS.

Setup Function

The :SYST:MAC command is fast, easy-to-use, and maintains an exact sequence of commands necessary to complete a setup that specify functions, channels, arming mode, and threshold voltages. The three setups below for functions TPD++, PW+, TT+ are stored in memory under ***SAV**e**1** and a **OP**eration **C**omplete command (*OPC?) queries a response from the DTS to status byte the MAV bit when ready.

The setup function writes important instructions to the DTS that will assign proper arming modes with channel information and threshold voltage values to accomplish a measurement. I recommend maintaining an exact sequence of commands as structured in the :SYST:MACRO command; in other words always follows a sequence of function, channel, arming, voltage; especially during the first basic measurement setup. Later you may want to change arming or count cycle or even threshold voltages and with a basic setup simple changes are sufficient.

```
void dts_setup( ) {
     ibwrt( dts_no,":SYST:MAC/TPD++/BOTH/AUT/STOP/  /0.200/0.200/", 44);
     ibwrt( dts_no,":SYST:MAC/PW+/ 1 /AUT/STOP/  /0.200/0.200/", 41);
     ibwrt( dts_no,":SYST:MAC/TT+/ 2 /AUT/STOP/20 80/ / /", 36);
     ibwrt( dts_no,"*SAV1; *OPC?", 12);
                    while (!(poll_status  & 0x 10))
                     {
                      ibrsp(dts_no,&poll_status);
                     }
          ibrd(dts_no, opcck, 5);
          printf (opc = %d \n", opcck);
 return;
 }
```

The save/recall commands stores all 10 functions, channels, voltages by **one** save command; therefore in this example the three setups TPD++, PW+, TT+ are all in memory under *SAV1. The query ***OPC?,** and serial polling of the status byte register for a MAV bit high assures the save is complete, and read **ibrd** response OPC = 1. Without the serial polling of the status byte, the ibrd could read "0" signifying the **SAV1** was not complete, so wait for the serial poll status then do the read. Table no.1 summarizes the three setups. Later, the three setups stored in memory can be recalled by the command *RCL1 in conjunction with the *ACRuire:RUN function command.

Function	Channel	Arming Enable Mode	Trigger	Voltage1
TPD++	Both	Auto Arm Stop	User	0.200
PW+	CH 1	Auto Arm Stop	User	0.200
TT+	CH 2	Auto Arm Stop	20% - 80%	Vol

Table No. 1 System Macro Command Setups

Measurement Commands

The measurement macro commands are very fast and easy to use. The quick measure command (:ACQuire:RUN PW+) not only recalls the total setup for PW+ including channel, voltage, and arming mode parameters; but also executes a measurement and returns the AVERage and JITTer values.

```
void dts_measpw( ) {
     ibwrt( dts_no,":ACQ:RUNPW+", 11);
                    while (!((poll_status  & 0x 11) = = & 0x 11))
                     {
                      ibrsp(dts_no,&poll_status);
                     }
          ibrd(dts_no, resp, 100);
          sscanf ( &resp, "%f%f", &average, &jitter);
          printf("average = %e \n", average);
          printf("jitter = %e \n", jitter);
 return;
 }
```

The need for additional commands and polling are eliminated by the (:ACQuire:RUN PW+) command. The command performs internally by the DTS firmware a setup recall for PW+ , burst measurement, and return measurement value. The program need only serial poll the status byte and wait for the **MAV** ready.

Data Types

A discussion of data types are necessary so the programmer can properly declare variables for the DTS. All voltage parameters are type real and declared as **double**. All burst measurements including AVERage and JITTer are type floating point and declared as **double**. All samples are type integer using type modifier long and declared as **long**.

Termination Characters

All serial stream data transfers must be null-terminated by a GPIB EOI or the NL character. The character for a NL (new line) is a ASCII 0A (hex), NULL is a ASCII 00 (hex), and EOI (end or identify) is a hardware line. This signifies when the GPIB transmission is terminated. Therefore, the parser automatically detects the character and terminates the data transfer.

Pulse Find

The DTS has **pulse find** capability to measure the **PEAK** voltages sine wave or the **voh/vol FLAT** square wave on the chan1 and chan2 signals are selected by command :SYST:WAVPEAK . To initiate a **pulse find** (:ACQ:LEV) command, the instrument will measure the signal high level and low level.

```
 void dts_levtt( ) {
          ibwrt (dts_no,":SYST:WAVPEAK; :TER?; :ACQ:LEV",30);
          ibwrt (dts_no,":TER?",5);
                    while (!(poll_status  & 0x 01))
                              ibrsp(dts_no,&poll_status);
           ibwrt (dts_no, ":CHANSTOP:MAX?", 14);
                    while (!(poll_status  & 0x 10))
                              ibrsp(dts_no,&poll_status);
          ibrd(dts_no, vmax_str, 20);
 return;
 }
```

This function is necessary in automatically determining the 20%-80% voltage percentage points for TT+ based on the signal vol/voh in the :ACQ:RUNTT+ application.
The :CHAN STOP:MAX? query command returns voh data. This function is necessary not only for determining amplitude verification of an output signal vol and voh, but also overshoot and/or undershoot parameters.

System Arming Macro

Be aware that all the examples are in the **auto arming mode** for auto arm on stop. The sequence control of startfirst or auto arm start first defines that CH 1(start) will always be measured before CH 2 (stop). If your application needs to use external arming with arming on pulse count; a fast and easy to use :SYST:ARM macro command is available. The :SYST:ARM command specifies trigger source, arming enable mode, channel arm, arm threshold voltages, arm slope, arm count, and has up to 10 arming arguments.

ibwrt(dts_no,":SYST:ARM/EXT/STARTFIRST/ARM1/ARM1/ / /POS/ /1/2/", 49);

The :SYST:ARM macro commands are setting up the EXTernal arming mode with enable sequence in the STARTFIRST mode. The ARM1 input will trigger off the start arming input treshold voltage (determined by the pulse find) and the arm1 slope will be POSitive and arm on start count of 1 and stop count of 2.

Program Example

The C++ Functions and commands of the DTS written in this application and incorporated in the test program are shown in table no. 2.

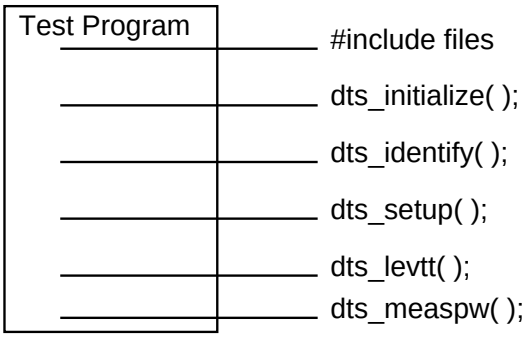

Table No. 2 C++ Functions in test program.

The arming enable modes are **auto** arm on **STOP** for both the measurement of the prop delay (TPD++) of channel 1 rising edge at 2.0vdc with respect to channel 2 rising edge at 2.0vdc with x10 probes; and the measurement of pulse width (PW+) of Channel 1 signal (see table no. 3 for waveform examples). The arming mode enables the measurement of " AUTO ARM ON STOP" or STOP and is one of four special DTS arming enable modes.

The arming mode **auto arm** on **stop** assures the stop pulse will be measured for rise time (TT+) at 20% to 80% of vol/voh. The DTS will measure any random event or signal and with the arming enable mode the user can control which event is measured first.

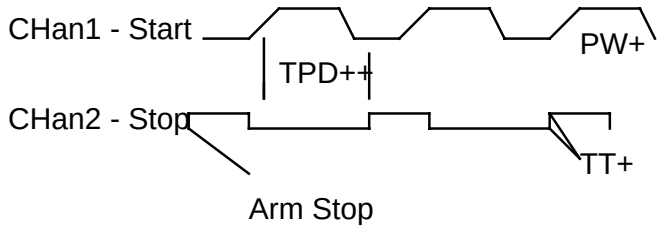

Table No. 3 Waveform examples.

Conclusions

The test commands of the DTS written in this application are for a ATE Test system using UNIX in a C/C++ environment with the NI-488.2M GPIB software driver.

The WAVECREST Digital Time Scope make the fast, accurate one-shot time interval measurements necessary for automated testing.

This paper will help the Test Engineer organize and structure his test program; so he can properly accomplish status reporting, serial polling, exchange protocol, convert measurement data formats, and utilize easy-to-use macros. The need to measure critical timing parameters on I.C.'s for propagation delays (TPD), pulse width (PW) duty cycle, and rise times (TT+) with 10pS. accuracy are demonstrated in the application discussed in this document. With the measurement goals and budget achieved; the user is able to write software to initialize the DTS via the GPIB bus, save measurement setups, execute measurements, and obtain critical device jitter data.

References

Mitchell Waite and Stephen Prata, New C Primer Plus, Second Edition, Sams Publishing, Carmel, IN.; 1993.

National Instruments Corp., NI-488.2 M Software Reference Manual, 1992.

Kerry Newcom, Streaming Data Speeds Up IEEE 488 Bus, Evaluation Engineering, Nokomis, Fl., June,1993.

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

Anatole Olczak, UNIX System V Quick Reference Guide Release 4, ASP, San Jose, Ca.; 1993.

IEEE-488 Interface Guide, Digital Time Scope DTS2070. Wavecrest Corporation, Edina, Mn.

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

200116-01 REV A