WAVECREST Corporation

PROGRAMMING THE WAVECREST DTS WITH THE WAVE.C DRIVER

Application Note No. 124

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

Contents

Introduction 4

The Wave.C Driver 4

Function Calls 5

An Example “C” Test Program 9

ANSI/IEEE-488.2 Standard 11

Data Types 11

Conclusions 11

References 11

Programming The Wavecrest DTS With The Wave.C Driver

Applications Note Number 124

Introduction

The *WAVECREST* Digital Time System (DTS) makes the fast, accurate real-time measurements and jitter analysis used in automated testing. When embedded in an ATE tester, the DTS uses the GPIB bus (IEE-488.2) to communicate commands and measurements to the processor. The WAVE.C driver was designed to provide a generic "C" driver that communicates with most testers and executes from a test program.

This application note explains the WAVE.C instrument driver used in an automated ATE environment to measure asynchronous events for frequency, period jitter, pulse width, propagation delay, rise time and fall time. It teaches how to:

1) Define and Use the " WAVE.C " calls.
2) Write a Test Program with a DTS Period and TPD measurement.
3) Call " C " functions that measure Frequency, Period Jitter, Pulse Width, Prop. Delay, Rise Time and Fall Time.
4) Return measurement data for comparison to limits for characterization or DUT/ device specifications.
5) Provide testing solutions to make timing measurements 1Ghz or less on synchronous or asynchronous events.

This paper provides essential software examples that guide the test engineer through the successful execution of programming instructions over the GPIB. The following WAVE.C examples use the calls and arguments of real test cases and show the DTS features..

The WAVE.C Driver

High performance Phase Lock Loop (PLL) Clock distribution network circuits (up, workstations, PC's, frequency dividers, frequency synthesizer) require measurement parameters of frequency, period, pulse width, rise time/fall time and propagation delays.

The WAVE.C driver was designed to perform testing using the WAVECREST DTS. The measurement calls described below are required PLL tests and are all available in the WAVE.C driver. The tester controls the embedded DTS with the WAVE.H include files linking the WAVE.C driver to the test program. The driver has specialized calls and arguments that perform the following:

1) Initializate the GPIB bus and Clear the DTS and Tester bus.
2) Identify the DTS.
3) Set up the DTS instrument function and channels.
4) Set up DTS arming, arming sequence control, and Nth event counters.
5) Execute and measure the DTS returned parameters of frequency, period, pulse width, rise time, fall time and propagation delays.
6) Execute and measure the DTS returned parameters of Pulse Find values of VOL and VOH.
7) Call GPIB status byte register serial polling routines for proper execution and efficiency.
8) Set up the DTS for sample size, window filters, display panels and any GPIB IEEE.2 command string.

FUNCTION Calls

The WAVE.C device driver can be divided into several categories. Each category is explained using the following examples of how the functions are called in a test program. Each function call has a set of arguments that are checked for validity. Only valid arguments are passed through the proper GPIB strings to execute the functions. Any invalid argument results in an error flag.

Initialize GPIB

```
BOOLEAN_t wave_initialize ( );
```

Identifiy the DTS

```
static BOOLEAN_t wave_identify( );
```

Setup the DTS Measurement

```
wave_setup_period(CHANNEL_1, PERCENT_50_50, NO_REF, NO_REF);
wave_setup_rise_time(CHANNEL_2,PERCENT_20_80,NO_REF, NO_REF);
wave_setup_fall_time(CHANNEL_1,PERCENT_80_20, NO_REF, NO_REF);
wave_setup_pulse_width(CHANNEL_1, PERCENT_50_50, NO_REF, NO_REF);
wave_setup_frequency(CHANNEL_1, PERCENT_50_50, NO_REF, NO_REF);
wave_setup_prop_delay(RISE_RISE,CHANNEL_12, PERCENT_50_50, NO_REF,
   NO_REF);
```

Setup the DTS Arming

```
wave_setup_arm ( AUTO_ARM, STOP_SEQ,  ,  ,  ,  ,  ,  ,  1,  2 ) ;
wave_setup_ext_arm (STOP_SEQ , ARM1 , NO_REF , POS , 1 ,  2 ) ;
```

Execute DTS Measurements

```
wave = wave_measure_all  (  );
wave = wave_measure (  );
```

Execute DTS Pulse Find Measurements

```
wave_pulse_find ( VOH    );
wave = wave_pulse_find_measure( VOH,CHANNEL_1 );
```

GPIB Status and Polling

```
static BOOLEAN_t wave_poll_meas( );
static BOOLEAN_t wave_poll_pf_trig( );
static BOOLEAN_t wave_poll_mav( );
```

Setup DTS Window Filters

```
wave_setup_filter ( ON, 1.00e-9, 5.100e-09  );
```

Setup the DTS Sample Size and Display

```
wave_setup_sample_size(  4 );
wave_setup_display( OFF );
```

Setup any DTS IEEE.2 Command string

```
strcpy ( gpib_cmds, " *CLS; :TL:EXE GR 1 " );
wave_setup_command( gpib_cmds );
```

Take DTS Offline at end of Testing Summary or Re-initialize

```
dts2070_offline ( );
```

Test Program Include Files

```
#include <ni_gpib.h>
#include "wave.h"
```

The **Initialize GPIB** example routine, "`BOOLEAN_t wave_initialize ( );`" will return a TRUE if the GPIB is connected and on-line. If the GPIB routine fails to initialize and receives an error, the result is a FALSE return. The next routine, "`static BOOLEAN_t wave_identify( );`", requests the DTS instrument to identify and respond with a return string "`WAVE DTS XXXX.XX.XX`." This string is very important as the first GPIB communication between the DTS and the tester.

In the **Set Up the DTS Measurement** example routine, "`wave_setup_period (CHANNEL_1, PERCENT_50_50, NO_REF, NO_REF );`" will set up the arguments period measurement function of channel 1 at the 50 % trigger level with no reference USER voltage values.

The `wave_setup_xxxxxx` syntax arguments for the function are:

wave_setup_xxxxxx	`xxxxxx = period, rise_time, fall_time, pulse_width, frequency`
Channel	`CHANNEL_1, CHANNEL_2, CHANNEL_12`

Pulse Find VOH	`PERCENT_50_50, PERCENT_20_80, PERCENT_10_90,`
	`PERCENT_80_20, PERCENT_90_10, PERCENT_20_20,`
	`PERCENT_30_30, PERCENT_80_80, PERCENT_70_70,`
	`PERCENT_40_40, PERCENT_60_60, PERCENT_90_90`

Start USER Input V	`NO_REF, +0.1500, -1.0000, + or - 1.100`
Start USER Input V	`NO_REF, +1.0000, -0.5000, + or - 1.100`

Notice the arguments are the same for all "`wave_setup_xxxxxx`" routines except for "`wave_setup_prop_delay (RISE_RISE, CHANNEL_12, PERCENT_50_50, NO_REF, NO_REF);`" The propagation delay has an extra argument for edge-to-edge type measurements.

The `wave_setup_prop_delay` syntax arguments for the function are:

wave_setup_xxxxxx	`xxxxxx = prop_delay`
Edge measurement	`RISE_RISE, RISE_FALL, FALL_RISE, FALL_FALL`
Channel	`CHANNEL_12`

Pulse Find VOH	`PERCENT_50_50, PERCENT_20_80, PERCENT_10_90,`
	`PERCENT_80_20, PERCENT_90_10, PERCENT_20_20,`
	`PERCENT_30_30, PERCENT_80_80, PERCENT_70_70,`
	`PERCENT_40_40, PERCENT_60_60, PERCENT_90_90`

Start USER Input V	`NO_REF, +0.1500, -1.0000, + or - 1.100`
Start USER Input V	`NO_REF, +1.0000, -0.5000, + or - 1.100`

The next catagory group example is the **Set Up any DTS IEEE.2 Command String**. In the example routine, "strcpy (gpib_cmds, "`*CLS; :TL:EXE GR 1`"); `wave_setup_command(gpib_cmds);`" will send a Command String to the DTS instrument.

The string copy command sends a `*CLS,` or clear the bus command, and a `:TL:EXE GR 1,` or Test List Execute Group number 1. Please refer to Wavecrest DTS IEEE488 manual (reference # 3) for GPIB commands examples.

The `wave_setup_command` syntax arguments for the function are:

GPIB IEEE.2 Any DTS command in the IEEE (GPIB) manual.

In the **Setup the DTS Arming** example routine, "`wave_setup_arm (AUTO_ARM, STOP_SEQ, , , , , , , 1,2);`" sets up the arming function for auto arm with a stop sequence control including start count 1 and stop count 2. Blanks or white space defaults do not apply. Please refer to Wavecrest Applications note #115 (reference # 5) for essential arming examples.

The `wave_setup_arm` syntax arguments for the function are:

Arming	`AUTO_ARM, EXT_ARM`
Arming Sequence	`STOP_SEQ, START_SEQ, STARTFIRST`
Start Arm	`ARM1, ARM2`
Stop Arm	`ARM1, ARM2`
Arm1 USER Input V	`NO_REF, +0.1500, -1.0000, + or - 1.100`
Arm2 USER Input V	`NO_REF, +1.0000, -0.5000, + or - 1.100`
Arm1 Slope	`POS, NEG`
Arm2 Slope	`POS, NEG`
Start Count	`1 to 256`
Stop Count	`1 to 256`

The **Setup the DTS Arming** example routine, "`wave_setup_ext_arm (STOP_SEQ , ARM1 , NO_REF , POS , 1 , 2);`" has fewer arguments because the default is always external arm. The setup arming function can only be EXT and the arguments are defined for stop sequence control with arm1 Pulse Find references, or no_ref, and arm1 triggers on the rising edge or pos with start count 1 and stop count 2. Please refer to Wavecrest Applications note #115 (reference # 5) for essential arming examples.

The `wave_setup_ext_arm` syntax arguments for the function are:

Arming Sequence	`STOP_SEQ, START_SEQ, STARTFIRST`
Start and Stop Arm	`ARM1, ARM2`
Arm1 or Arm2 USER Input V	`NO_REF, +0.1500, -1.0000, + or - 1.100`
Arm1 or Arm2 Slope	`POS, NEG`
Start Count	`1 to 256`
Stop Count	`1 to 256`

The **Execute DTS Measurements** example routine `wave = wave_measure_all ( );` has no arguments because the measured results will be measurement parameters of mean/average, RMS jitter, maximum and minimum values. The returned string will be a floating point double with a white space delimiter.

The second example routine `wave=wave_measure( );` also has no arguments because the measured results will be measurement parameters of Mean/Average, RMS Jitter. The returned string will be a floating point double with a white space delimiter.

The next catagory group example, **Execute DTS Pulse Find Measurements**, measures the amplitude of the input signals. The signals can be sine wave or square wave or even dc levels. Pulse find will measure the amplitude with the routine example "`wave_pulse_find (VOH);`", and in the argument, VOH is a square wave burst of events. This function executes a measurement and returns no value. Many of the setups use the 50% value of this measurement which means the user is not required to return this value.

The wave_pulse_find syntax arguments for the function are:

Pulse under Measement `PEAK, VOH`

The second example routine returns a value. The `wave=wave_pulse_find_measure (VOH, CHANNEL_1);`" example returns a value for VOH measured on channel 1. All voltage parameters are real and declared as double. The user may require to test the VOH value against a limit or specification.

The `wave_pulse_find_measure` syntax arguments for the function are:

Pulse Find Result Channel

```
VOH, VOL, VOH_VOL , BOTH
CHANNEL_1, CHANNEL_2, CHANNEL_12, BOTH
```

The next catagory group example are GPIB serial polling functions.

In the **GPIB Status and Polling** example routine, "`wave_poll_meas ( );`" will return a TRUE if the measurment is complete. The next example routine, "`wave_poll_pf_trig( );`", will return a TRUE if the pulse find amplitude measurement is complete. In the last example routine, "`wave_poll_mav( );`" will return a TRUE if the status byte regisiter MAV bit is a "1" which means a message is available. These functions are used during initialization and other routines when querys are used in the test program.

The next catagory group example are the **Setup DTS Window Filters.**

In the example routine, "`wave_ setup_filter(ON, 1.00e-9, 5.100e-09);`" will turn on the window filter and setup the filter parameters to accept events that are within 1.0nS upto 5.1ns. The filter is used to exclude measurements or distributiuons.

The `wave_pulse_find_measure` syntax arguments for the function are:

Window Filter `ON, OFF`
Filter Min Value `-2.5 Sec. to + 2.5 Sec.`
Filter Max Value `-2.5 Sec. to + 2.5 Sec.`

The next catagory group example is the **Setup the DTS Sample Size and Display.** This routine, "`wave_setup_sample_size(4);`", will set the sample size to 4. The sample size is declared as a long integer.

The `wave_setup_sample_size` syntax arguments for the function are:

Sample Size `1 - 1,000,000`

In the example routine, "`wave_setup_display(OFF);`" will turn the front panel off. This is a throughput enhancement since there is overhead time to refresh the front panel.

The `wave_setup_display` syntax arguments for the function are:

Display Panel `ON, OFF`

The next catagory group example is the **Take DTS Offline at end of Testing Summary or Re-initialize.** In the example routine, "`dts2070_offline ( );`" sends an offline command string to the GPIB and disables the bus. To enable the GPIB bus the user must initialize the DTS instrument with the `wave_initialize ( )` function.

An Example “ C “ Test Program

The example function below is a C++ test program. The program code will declare variables, call functions to initialize the GPIB and DTS, call a function to turn the DTS display off, call a function to execute a pulse find, call a function to set up a TPD measurement from rising edge to rising edge, set up external arming, execute the TPD measurement, call a function to set up a sample size of 4, set up to measure period, set up auto arming, exectute a pulse find and execute a period measurement.

At the end of testing or at a bin summary, the call to return the DTS display to ON is executed. Having the DTS front panel in display mode allows manual measurements and operator intervention. See the following example C++ program.

Function Test Program

```
/*  include files and header files */
#include <ni_gpib.h>
#include “wave.h”

/* test program varables declarations */
static int dts_gpib = 5;
static char gpib_cmds[200];
int flag;

BOOLEAN_t  gpib_enabled = FALSE;

WAVE_RESULT_TYPE *wave_measure( );
WAVE_RESULT_TYPE *wave_pulse_find_measure( );

/*********************************************************************/
/*****               GPIB initialize WAVE DTS2070 TEST            *****/
/*********************************************************************/
void initialize_tester( )
{

/* ***** WAVE initialize GPIB *****  */

if(wave_initialize( ) == FALSE) exit( );
else gpib_enabled = TRUE;

/* ***** WAVE Front Panel OFF *****  */
wave_setup_display(OFF);
}
/*********************************************************************/
/******                        FUNCTIONS                          *****/
/*********************************************************************/

/* ***** WAVE setup and measurements for PF Pulse Find VOH VOL ***** */
void wave_PF_setup( )

{
        wave_pulse_find(PEAK);
 }

/* **** WAVE setup and measurements Prop. Delay Tpd++  **** */
void wave_TPD1( )

{
wave_setup_prop_delay(RISE_RISE,CHANNEL_12,PERCENT_50_50,NO_REF,NO_REF);
wave_setup_arm(EXT_ARM,STOP_SEQ,ARM1,ARM1,0.5000,NO_REF,POS,NO_SLOPE,1,1
);
wave = wave_measure( );
log_printf("\nTPD++ Measurement: %sS\n",eng_n(wave-> measure1));
log_printf("\nTPD++ JITTER Measurement: %sS\n",eng_n(wave-> measure2));
}
```

```
/*********************************************************************/
/*** This function is called from main and is used to test device  ***/
/*********************************************************************/
void test_device()
{
WAVE_RESULT_TYPE        *wave;
char reply[20];
int s,size,timeset;
int i,j,k;
float vdd,VOLT_REG;
double testtimes[20];
double result_time;
int             color;
double          current[NUM_OF_PINS];
float           curr;
float res[20];

/*********************************************************************/
/****              WAVE DTS2070 SETUP PERIOD AND TPD++             ****/
/*********************************************************************/
/*    WAVE DTS2070 setup stuff  */
if (gpib_enabled == FALSE) exit();

/* sample size = 4 */
wave_setup_sample_size(4);

/* setup period channel 1 50% of pulse find value  */
wave_setup_period ( CHANNEL_1,PERCENT_50_50,NO_REF,NO_REF );

/* setup auto arm with stop sequence start count 1 and stop count 2 */
wave_setup_arm ( AUTO_ARM, STOP_SEQ,  ,  ,  ,  ,  ,  ,  1, 2 ) ;

/* execute a pulse find measurement   */
wave_pulse_find ( PEAK );

/* request the measurement VOH for channel 1 */
wave = wave_pulse_find_measure ( VOH,CHANNEL_1 );

log_printf("\nPULSE FIND VOH Measurement: %sV\n",eng_n(wave-> voltage1));

wave = wave_measure ( );
log_printf("\nPERIOD Measurement: %sS\n",eng_n(wave-> measure1));
log_printf("\nPERIOD JITTER Measurement: %sS\n",eng_n(wave-> measure2));

/*********************************************************************/
/*test1 tests DUT TPD for wave dts2070 connected to DUT relay matrix */
/*********************************************************************/
test1:
     close_fixture_relays(tpd_io16_io5,0.01);
      loop_func_pat ();
             test_name("FUNC_1");
      wave_TPD1( );
      func_stop();
/*********************************************************************/
/************************ END OF TESTS ***************************/
/*********************************************************************/
void shutdown_device()

{
/*   this function call returns the DTS2070 front panel display on   */
 wave_setup_display ( ON );

}
```

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 and listen to all ATE systems controllers compliant to these standards. There are basic fundamental GPIB software commands a user must perform or the controller and instrument will not communicate. Please refer to Wavecrest IEEE manual (reference # 3) for essential software examples to initialize and setup the DTS with C/C++ software commands and necessary function calls to the GPIB NI driver software.

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, JITTer, RANGe, MINimum, MAXimum, and JITTer are type floating point and declared as double.

Conclusions

The DTS embedded in a ATE Tester can perform high speed time measurement in automated environments using the generic " WAVE.C " driver.

As discussed in this application note the DTS is used to test many high speed timing parameters including frequency, period, pulse width, rise and fall times, and propagation delay. The user can develop a test program and use the WAVE.C driver calls to initialize, setup, and measure and compare many vital parameters. Another important benefit of the DTS is embedded in the ATE Tester creates the capability to perform measurements up to 1Ghz with 10pS accuracy on any high performance Phase Lock Loop (PLL) Clock distribution circuits. Whether the tests are production or characterization of device jitter, the WAVE.C can provide the critical calls to functions that setup and execute timing measurements using the DTS.

Finally, in the application example " C++ " test program, the Test Engineer learned to organize and structure his test program using the WAVE.C function calls. This paper showed the user how to develop the test setups

References

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

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

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

4. Wavecrest Applications Note No. 114; Achieving 30ps. Accuracy . Wavecrest Corp., Edina, Mn.

5. Wavecrest Applications Note No. 115; Arming the Wave DTS. Wavecrest Corp., 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

200124-01 REV A