cadence®

Virtuoso® UltraSim Waveform Interface Reference

Product Version 18.1
January 2019

Cadence Design Systems, Inc. (Cadence), 2655 Seely Ave., San Jose, CA 95134, USA.

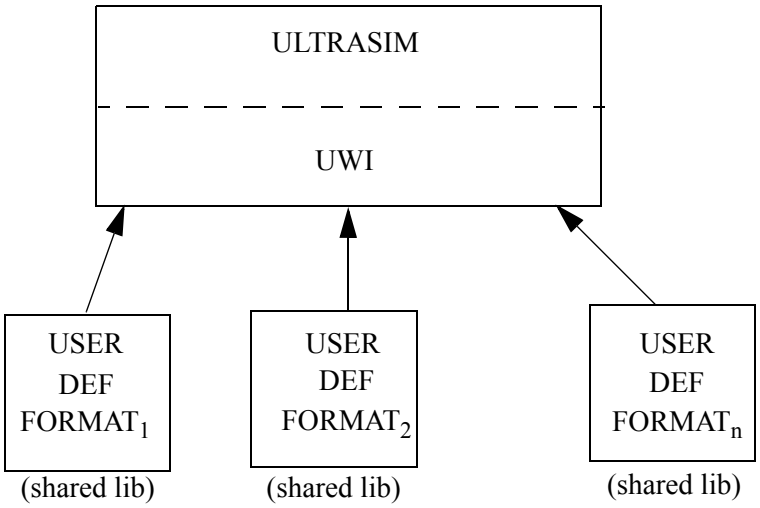
Contents

Virtuoso UltraSim Waveform Interface

The Virtuoso® UltraSim™ waveform interface (UWI) lets you write Virtuoso UltraSim probe data and read probe data into the Virtuoso UltraSim simulator. The current application programming interface (API) supports writing Virtuoso UltraSim simulation data to user-defined formats. This document describes creating Virtuoso UltraSim waveform outputs in user-defined formats.

The Virtuoso UltraSim waveform interface is comprised of a set of functions that enables simulation data to be written in a variety of formats (see Figure 1-1).

Figure 1-1 UWI Overview



You create source files (C-source files) containing the API function definitions for each output format specified in the netlist. You can specify more than one format in the netlist and, for each format, you should define the APIs in a separate `.C` source file. Dynamic libraries are created from these source files and are linked to the Virtuoso UltraSim simulator during run time.

UltraSim Waveform Interface Directory Structure

The Virtuoso UltraSim waveform interface functions are linked to the Virtuoso UltraSim simulator through dynamically shared libraries. The shared libraries contain user-provided definitions to Virtuoso UltraSim waveform API calls.

To minimize the effort required for creating these libraries, auto-build scripts and makefiles are included in the `ultrasim/` directory. Figure 1-2 shows the complete waveform interface directory structure.

Figure 1-2 UWI Directory Structure

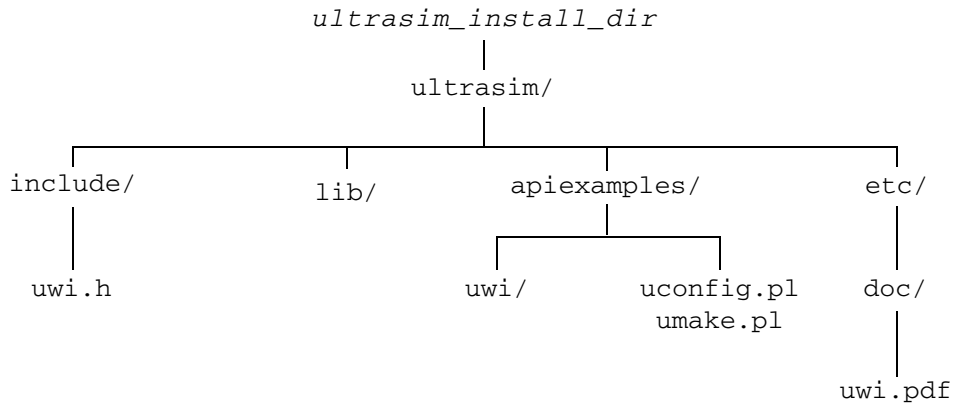


Table 1-1 UWI Directories and Files

Directory and Filenames	Descriptions
`etc/doc`	Virtuoso UltraSim waveform interface documentation.
`/etc`	Automated scripts for building library.
	`uconfig` and `umake` for automating the process of building the shared Virtuoso UltraSim waveform interface library.
`/apiexample`	Example source files, config file, and `makefile.file`.
`/include`	Header file for Virtuoso UltraSim waveform interface structures and function prototypes.
`/lib`	Object files for building shared library.

uwi_lib

```
.usim_opt
     [ wf_format = formatName ]
     uwi_lib = libraryPath
```

Arguments

formatName	Specifies the output waveform format. Different output waveform formats can be generated using `wf_format` statements. Set `wf_format` in the `uconfig.pl` file. Valid Values: `fsdb`, `sst2`, `psf`, `psfascii`, `wdf`, `psfxl`, and user-defined formats. Default: `sst2`
libraryPath	Specifies the shared libraries for the requested formats. All formats require a `uwi_lib` statement, except for `sst2` (default), `psf`, `fsdb`, `psfascii`, `wdf`, and `psfxl`.

If waveforms need to be generated in `sst2` format, specify this information in the netlist along with the other formats.

Example

```
.usim_opt wf_format = xydb
.usim_opt uwi_lib = ./libXYDB.so
```

Generates all signals specified in the probe statements in `xydb` format. The shared library contains definitions of API functions to create the user-defined output format.

Note: The Virtuoso UltraSim simulator generates signals in user-defined formats and, if required for postprocessing, in `sst2` formats. Postprocessing occurs when the netlist contains `.measure` statements.

UltraSim Waveform Interface APIs

This section describes the Virtuoso UltraSim waveform interface API syntax and functionality. The data structures mentioned in this section are specified in the `uwi.h` file.

uwi_WfIntDef

```
struct uwi_WfIntDef {
     uwi_StreamHandle (*open) (const uwi_Setup* );
     uwi_WfHandle (*defineWf) (uwi_StreamHandle, const uwi_WfDefinition* );
     void (*defineAliasName) (uwi_StreamHandle stream, uwi_WfHandle wfHdl,
                              const struct uwi_WfDefinition *aliasWfDef);
                              /* Add the alias name for the current waveform. */
     int (*endDefineWfs( uwi_StreamHandle );
     int (*addDWfPoint) ( uwi_StreamHandle, uwi_WfHandle, uwi_Logic, double t);
     int (*addAWfPoint) (uwi_StreamHandle, uwi_WfHandle, double, double);
     int (*flush)( uwi_StreamHandle );
     int (*close) (uwi_StreamHandle);
     int (*resetXcoord)(uwi_StreamHandle);
     char* (*getErrMsg) ( );
     const char* format;
     }
```

Description

Registers the public functions of a waveform library with the Virtuoso UltraSim simulator. The function is called on each shared library after loading. The function returns a pointer to a `uwi_WfIntDef` structure containing pointers to the API functions. The user defines the functions and assigns them to the members of this structure.

Note: The memory to the structure is allocated by the user and should not be freed until the Virtuoso UltraSim process is terminated. The intention is that each shared library contains a static instance of such a structure.

The `uwi_register` function is required for every library. The Virtuoso UltraSim simulator, in addition, has the following requirements on the function handlers returned. The functions `open` and `defineWf` are mandatory.

The format character string within the `uwi_WfIntDef` structure identifies the provided waveform format. This string needs to match the `wf_format` string in the netlist to activate the provided waveform output.

Example

```
uwi_StreamHandle open( const uwi_Setup* initData )
```

This function is called whenever a new file stream in the user-defined output format needs to be opened. `uwi_StreamHandle` associates the waveforms with the format written.

```
typedef void* uwi_StreamHandle;
```

This function is called by the Virtuoso UltraSim simulator with a `uwi_Setup` argument. The argument contains the description of the global simulation information valid for the current output being written.

Note: On return of the function, the Virtuoso UltraSim simulator deletes or reuses the memory of the structure argument passed to the function.

uwi_Setup

```
struct uwi_Setup {
     const char* filename;
     uwi_AnalysisType analysis;
     int alterIter;
     int ageIter;
     double temp;
     double tRes;
     }
```

Description

The information provided in the `uwi_Setup` structure is intended to provide a precise description of the overall simulation run associated with this stream. *`filename`* is the name of the SPICE netlist and the analysis field is defined as

```
enum uwi_AnalysisType {TRAN, DC, NOISE, AC}
```

`alterIter`, `ageIter`, `temp`, and `tRes` provide additional information in case of more complex simulation runs. In general, output systems can take advantage of this information beginning with better error checking and output filename creation up to more efficient data structures. `tRes` gives the time resolution of the x axis specified in the `usim` options. The default is 1ps.

Example

You can replace *`filename`* in `uwi_Setup` with your own output file and then open that to dump the waveform or other information. If you open a file for writing and you want it to be used as the stream, that file handle must be returned by

```
uwi_WfHandle defineWf( uwi_StreamHandle stream, const uwi_WfDefinition* wfDef )
```

This function creates a handle for the given waveform definition in the given stream. The signal name, scope of the signal, units represented, and type of the signal are passed through by the uwi_WfDefinition structure. The function returns a pointer to `uwi_WfHandle` if successful. The waveform handle is subsequently used while calling the function to write Virtuoso UltraSim simulation data.

```
typedef void* uwi_WfHandle;
```

uwi_WfDefinition

```
struct uwi_WfDefinition {
     const char* wfName;
     const char** scopeName;
     const char* units;
     uwi_WfType wfType;
     }
```

Description

`uwi_WfDefinition` is defined as a structure containing all signal-specific information. The instance of this structure is controlled by the Virtuoso UltraSim simulator and is not guaranteed to exist during the call to the `defineWf` function.

The individual field descriptions are as follows:

- `wfName` represents the actual signal name
- `scopeName` is a null-terminated `char*` array

  Each entry in this array represents the part of the hierarchical path to this signal, starting with the top level.
- `units` identifies the physical unit of this signal (either a V for voltage or A for current)
- `uwi_WfType` is an `enum` defined as

  ```
  enum uwi_WfType {ANALOG,DIGITAL}
  ```

The type of a distinguishes between continuous time signals and discrete signal values.

Note: At the return of the function, the Virtuoso UltraSim simulator deletes or reuses the memory allocated to the structure.

Examples

```
int endDefineWfs( uwi_StreamHandle stream )
```

This function indicates the completion of definition of all waveforms to be added to the database. This optional function can be ignored by setting the function pointer in the `uwi_WfIntDef` structure to `null`.

```
int addDWfPoint( uwi_StreamHandle stream, uwi_WfHandle wfHandle, uwi_Logic val,
double t )
```

This function sets the digital value of the waveform in a stream, pointed to by `wfHandle`. The x axis value is passed through the argument `t` (for DC, `t` is ignored). The digital value passed through the argument value is one of the enumerated types defined in `uwi_Logic`. If successful, the function returns `0`.

```
enum uwi_Logic {IN, OUT, X, Z}
int addAWfPoint(uwi_StreamHandle stream, uwi_WfHandle wfHandle, double val, double
t)
```

This function sets the analog value of a waveform in a stream pointed to by `wfHandle` for the time `t`. If successful, the function returns `0`.

```
int flush(uwi_StreamHandle stream)
```

This function is called when the Virtuoso UltraSim simulator requires the flushing of the waveform data corresponding to the `uwi_StreamHandle`. If successful, the function returns `0`.

```
int close(uwi_StreamHandle stream)
```

This function closes the stream handle instance referred to by `stream`. This function is called once for each stream opened. The function returns `0` if successful. It is a good practice to close the open streams. That would avoid any conflict during output process.

```
int resetXCoord(uwi_StreamHandle stream)
```

This function resets the x coordinate of the database to time 0. All signals that are written after this function would start from time zero.

```
char* getErrMsg()
```

This function returns a textual description of the last error that occurred in one of the API functions. Depending on the importance of an error within the Virtuoso UltraSim flow, the textual message might be printed to the output.

Note: Other functions and variables defined in the `uwi.h` header file are used internally by the Virtuoso UltraSim simulator and can be ignored.

Building a Shared Library

This section describes how to use the scripts provided to build the shared library.

To build a shared library,

1. Create a directory and create the `.c` files containing the function definition of the APIs.
2. Run `uconfig.pl` in the directory containing the source files (`.c` files) by typing

```
uconfig.pl uwi [ -P:64 ]
```

(Use `-P:64` for 64-bit applications.)

A makefile to create the library is generated.

3. Compile to code using the generated makefile and create the shared library by typing

```
umake.pl
```

Windows NT users should consider the following:

- The system variable `CELESTRT_HOME` should be set correctly to the `ultrasim/` home directory.
- The compiler used in the scripts to build the library is the Visual C++ compiler. Set the Windows system variables `INCLUDE`, `PATH`, and `LIB` to specify the include files, the Visual C++ path, and the library before using the scripts to build the library.

  For example, if your Visual C++ is installed at `C:\Program Files\Microscoft Visual Studio`, the Windows system variable `INCLUDE` should include the directory `c:\Program Files\Microsoft Visual Studio\vc98\include`, `PATH` should include the directory `c:\Program Files\Microsoft Visual Studio\vc98\bin`, and `LIB` should include `c:\Program Files\Microsoft Visual Studio\vc98\lib`.

Note: To use the Virtuoso UltraSim waveform interface, you need to recompile the UWI header file (MMSIM 6.0 USR1 and newer releases).

UltraSim Waveform Interface Example

This example illustrates the use of these APIs to dump a waveform in a simple ASCII format.

```
/************************************************************
This is a simple program to explain the UWI interface. The main
purpose of this code is to demonstrate the UWI interface through
a very simple example.
************************************************************/

#include <stdio.h>
#include <string.h>
#include "uwi.h"

/************************************************************
This structure defines a stream in terms of a file pointer where data will be
stored. It also has arrays for data, digital time, analog time and digital value
to store waveform data
************************************************************/

typedef struct wfInfo
{
    int  index;
```

```
    double time;
    double val;
    short isDigital;

} sigData;

struct wfFormat
{
    FILE* fp;/*   Defines output stream    */
    char* sigName[100]; /*   Stores the name of the signals    */
    sigData signal[1000]; /*   Stores signal name starting and index     */
    int sigCount; /*   Stores the index of the signal Name     */
    int dataCount;/*   Stores the number of data points to be flushed */
};

static struct wfFormat format; /* Creates static instance of struct to store
waveform */

/* Function delclaration to convert and integer to corresponding string value {0,
1, x, z}*/
char* getLogicStr(int val);

char* getLogicStr(int val)
{
    if(val == 0)
    return "0";
    else if(val ==1 )
    return "1";
    else if(val == 2)
    return "x";
    else if(val == 3)
    return "z";
    else
    return "";
}


/* This defines the open function */
uwi_StreamHandle open(const struct uwi_Setup* setup)
{
    char name[1000];

    /* Initialize the data-structure wfFormat */
    format.fp         = NULL;
    format.sigCount   = 0;

    strcpy(name, setup->fileName);

    /* create a unique file name depending upon the analysis type */



    if(setup->analysis == DC)
    strcat(name, "_DC.saf");
    else if(setup->analysis == TRAN)
    strcat(name, ".saf");
    else if(setup->analysis == NOISE)
    strcat(name, "_NOISE.saf");
    else if(setup->analysis == AC)
    strcat(name, "_AC.saf");
```

```
    /*open the file*/
    format.fp = fopen(name, "w+");

    /* Print the header information*/
    fprintf( format.fp,"\tSignal\t\tTime\t\tVal\n");

    return format.fp;
}

/* Defining the waveform handle function */
uwi_WfHandle defineWf(uwi_StreamHandle strHandle, const struct uwi_WfDefinition*
wfDef)
{
    int index, sigSize = 0;

    /* Allocate memory for the signal */
    if( wfDef->wfName )
    sigSize = sizeof( wfDef->wfName );

    if( sigSize )
    format.sigName[format.sigCount] = (char *)malloc(sigSize);
    else
    return NULL;

    /* Copy the signal name to the struct defined at the start of the program */
    strcpy(format.sigName[format.sigCount], wfDef->wfName);

    /* increment the index of the name array */
    index = ++format.sigCount;

    /* Return the index as Waveform Handle */
    return (uwi_WfHandle)index;


}

/* Define the flush function */
int flush(uwi_StreamHandle stream)
{
    int i;
    sigData  sig;
    char*  name;

    for(i = 0; i< format.dataCount; ++i)
    {
    sig = format.signal[i];
    name = format.sigName[sig.index-1];
    if( !sig.isDigital )
    fprintf(stream, "\t%s \t%e  \t%e\n", name, sig.time, sig.val);
    else
    {
    char* lStr = getLogicStr((int)sig.val);
    fprintf(stream, "\t%s \t%e  \t%s\n",name, sig.time, lStr );
    }
    }
    return 0;
}

/* Defines the close API */
int close(uwi_StreamHandle stream)
```

```
{
    int i;
    fclose((FILE*)stream);
    for( i = 0 ; i < format.sigCount ; ++i )
    free( format.sigName[i] );
    return 0;
}

/* Defines the addDWfPoint API */
int addDWfPoint(uwi_StreamHandle stream, uwi_WfHandle wfHandle, enum uwi_Logic
val, double t)
{
    /* Add the digital value and the corresponding time into the array of the struct
    */
    long index = (long)(wfHandle);

    format.signal[format.dataCount].time      = t;
    format.signal[format.dataCount].val       = (double)val;
    format.signal[format.dataCount].index     = index;
    format.signal[format.dataCount].isDigital = 1;
    ++format.dataCount;

    return 0;
}

/* Defines addAWfPoint API */
int addAWfPoint(uwi_StreamHandle stream, uwi_WfHandle wfHandle, double val, double
t)
{
    long index = (long)(wfHandle);

    format.signal[format.dataCount].time      = t;
    format.signal[format.dataCount].val       = val;
    format.signal[format.dataCount].index     = index;
    format.signal[format.dataCount].isDigital = 0;
    ++format.dataCount;

    return 0;
}


/* Finally register all the functions defined above with uwi_register() API */
struct uwi_WfIntDef* uwi_register()
{
    /* Defines a static struct of the waveform interface definition */
    static struct uwi_WfIntDef wfIntDef;

    /* Assigns all the user defined functions to the members of the waveform
    interface object */
    wfIntDef.open = open;
    wfIntDef.defineWf = defineWf;
    wfIntDef.endDefineWfs = NULL;
    wfIntDef.addDWfPoint = addDWfPoint;
    wfIntDef.addAWfPoint = addAWfPoint;
    wfIntDef.flush = flush;
    wfIntDef.close = close;
    wfIntDef.resetXCoord = NULL;

    /* the format is SAF the user needs specify the same format in the netlist */
    wfIntDef.format = "SAF";
```

```
    return &wfIntDef;
}
```