cadence®

Spectre® Unified Waveform Interface Reference

Product Version 23.1
June 2023

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

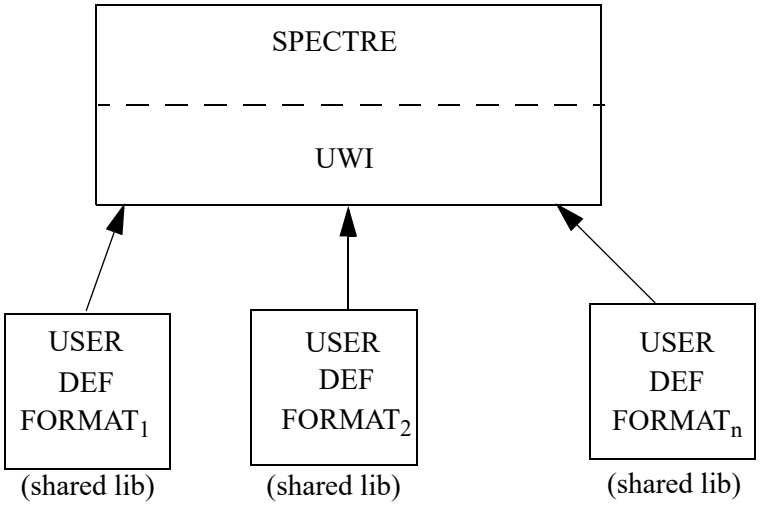
Contents

Spectre Unified Waveform Interface

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

The Spectre UWI 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 Spectre UWI Overview



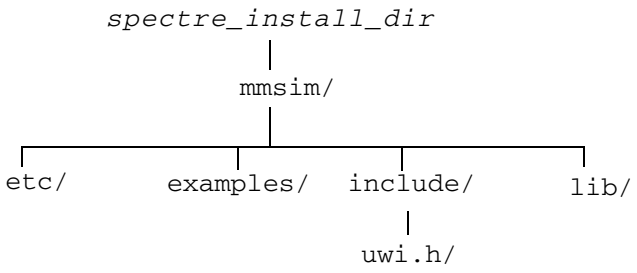
You create source files (C-source files) containing the API function definitions for each output format specified in the netlist. You can specify more than one format in the netlist and, for each format, you should define the APIs in a separate `.C` source file. Dynamic libraries are created from these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

Spectre UWI Directory Structure

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

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

Figure 1-2 UWI Directory Structure



uwi_lib

```
opt options
     [ uwifmt = formatName ]
     uwilib = libraryPath
```

Arguments

formatName	Specifies the user-defined output format. To specify multiple formats, use `:' as a delimiter. The option is valid only when the waveform format is defined as `uwi`.
libraryPath	Specifies the absolute path to the user-defined output format library. This option is used along with `uwifmt`. Use `:' as a delimiter to specify more than one library.

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

Example

```
opt options uwifmt = xydb
opt1 options uwilib = ./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 Spectre simulator generates signals in user-defined formats and, if required for postprocessing, in `sst2` formats. Postprocessing occurs when the netlist contains `.measure` statements.

Spectre UWI APIs

This section describes the Spectre UWI API syntax and functionality. The data structures mentioned in this section are specified in the `uwi.h` file.

uwi_WfIntDef

```
struct uwi_WfIntDef {
    void   *( *open )( const struct uwi_Setup *initData );
    void   *( *defineWf )( uwi_StreamHandle stream, const struct
                           uwi_WfDefinition *wfDef );
    int ( *endDefineWfs )( uwi_StreamHandle stream );
    int ( *addDWfPoint )( uwi_StreamHandle stream, uwi_WfHandle wfHdl,
                          enum uwi_Logic value, double time );
    int ( *addAWfPoint )( uwi_StreamHandle stream, uwi_WfHandle wfHdl,
                          double value, double time );
    int ( *addAWfComplexPoint )( uwi_StreamHandle stream, uwi_WfHandle wfHdl,
                                 double vReal, double vImage, double time );
    int ( *flush )( uwi_StreamHandle stream );
    int ( *flushNow )( uwi_StreamHandle stream );
    int ( *close )( uwi_StreamHandle stream );
    char *( *getErrMsg )();
    int ( *resetXCoord )( uwi_StreamHandle stream );
    char *format;
    int ( *addTapIntervalData ) ( uwi_StreamHandle stream, uwi_WfHandle tapHdl, int
                                  intervalIndex, double avgValue, double peakValue,
                                  double rmsValue );
    int ( *addStaticCapacitance ) ( uwi_StreamHandle stream, uwi_WfHandle tapHdl,
                                    double tapCap, double devRes, double
                                    devOutLoad );
    int ( *limitFileReached ) ( uwi_StreamHandle stream, double maxFileSizeInBytes,
                                int currBufSizeInNumberOfPoints );
    void ( *updateFileSize ) ( uwi_StreamHandle stream, int sizeUpdated );
    void ( *defineAliasName ) ( uwi_StreamHandle stream, uwi_WfHandle wfHdl,
                                const struct uwi_WfDefinition *aliasWfDef );
    void ( *setMaxFileSize ) ( double maxFileSizeInBytes );
};
```

Description

Registers the public functions of a waveform library with the Spectre 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 Spectre 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 Spectre 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 Spectre 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 Spectre simulator deletes or reuses the memory of the structure argument passed to the function.

uwi_Setup

```
struct uwi_Setup {
    const char             *fileName;
    const char             *fileNameExt;
    enum uwi_AnalysisType   analysis;
    double                  temp;
    int                     runNum;
    int                     alterIter;
    int                     ageIter;
    double                  tRes;
    double                  tStop;
    const char             *rawDir;
    int                     isDouble;
    const char             *netName;
    const char             *amsWaveName;
    int                     tapNumber;
```

```
    double                  clockCycle;
    int                     numCycles;
    int                     numIntervals;
    int                     avgMethod;
    int                     peakMethod;
    int                     rmsMethod;
    int                     splitNum;
    int                     nTotalHarm;
    int                     iCurrentHarm;
    int                     envPostProcess;
    double                  tStart;
    char                   *title;
    unsigned int            distrFileNum;
};
```

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 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 the Spectre simulation data.

```
typedef void* uwi_WfHandle;
```

uwi_WfDefinition

```
struct uwi_WfDefinition {
    const char         *wfName;
    const char        **scopeName;
```

```
    const char          *units;
    enum 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 Spectre 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 Spectre 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 Spectre 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 Spectre 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 Spectre simulator and can be ignored.

Building a Shared Library

Assuming that SPECTRE_INSTALL is the root of the SPECTRE installation, the path to the `uwi.h` file is `${SPECTRE_INSTALL}/tools/mmsim/include/`. When adding this path to the compile line using the `-I` option, you can add the path to the `mmsim/include` directory, rather the path to the plug-ins directory, as follows:

```
% gcc -fPIC -I${MMSIM_INSTALL}/tools/mmsim/include -o myplugin.o -c myplugin.c
```

No Cadence libraries need to be linked to the final plug-in DLL. To create the plug-in, all object files should be compiled with the appropriate PIC option and then linked together into the DLL.

The following example uses `gcc` to create the shared library.

```
% gcc -shared -o libmyXYDB_sh.so myuwi.o % cp libmyXYDB_sh.so ~/plugins/`cds_plat`
```

Spectre UWI 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. This file is not recommended to be used as
a general format for all spice netlist. It has several limitations
and would fail for other spice netlist if used without modifying
the code
*****************************************************************/

#include <stdio.h>
#include <string.h>
#include <stdlib.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
*****************************************************************/

#define MAX_SIG_COUNT  100000
#define MAX_DATA_COUNT 1000000
#define MAX_CHAR       1000

typedef struct wfInfo
{
    int  index;
    double time;
    double val;
    double imag;
    short isDigital:1;
    short isComplex:1;
} sigData;

struct wfFormat
{
    FILE* fp;               /* Defines output stream                */
    char* sigName[MAX_SIG_COUNT]; /* Stores the name of the signals        */
    sigData signal[MAX_DATA_COUNT]; /* 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)
{
    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 openSAF(const struct uwi_Setup* setup)
{
    char name[MAX_CHAR];
    int STR_LEN = MAX_CHAR - 10; /* Max length of the file name that would be *
                                  * copied. Any name greater than that would be
                                  * truncated    */

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

    snprintf( name, STR_LEN, "%s/%s", setup->rawDir, setup->fileName  );

    /* create a unique file name depending upon the analysis type */
    if(setup->analysis == UWI_DC)
        strcat(name, "_DC.saf");
    else if(setup->analysis == UWI_TRAN)
        strcat(name, ".saf");
    else if(setup->analysis == UWI_NOISE)
```

```
        strcat(name, "_NOISE.saf");
    else if(setup->analysis == UWI_AC)
         strcat(name, "_AC.saf");
    else if(setup->analysis == UWI_INFO)
         strcat(name, "_INFO.saf");
    else
        // Not supported analysis
        return NULL;

    /* 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 defineWfSAF(uwi_StreamHandle strHandle, const struct
                                                    uwi_WfDefinition* wfDef)
{
    int index, sigSize = 0;

    /* Allocate memory for the signal */
    if( wfDef->wfName )
        sigSize = strlen( wfDef->wfName );
    int it;
    if ( wfDef->scopeName && wfDef->scopeName[0] != 0 ) {
        for (it = 0; wfDef->scopeName[it] != 0; ++it ) {
            sigSize += 1 + strlen(wfDef->scopeName[it]);
        }
    }


    if( sigSize && format.sigCount < MAX_SIG_COUNT )
        format.sigName[format.sigCount] = (char *)malloc(sigSize+1);
    else
        return NULL;

    /* Copy the signal name to the struct defined at the start of the program */
```

```
    if ( wfDef->scopeName && wfDef->scopeName[0] != 0 ) {
        format.sigName[format.sigCount][0] = '\0';
        for (it = 0; wfDef->scopeName[it] != 0; ++it ) {
            strcat(format.sigName[format.sigCount], wfDef->scopeName[it]);
            strcat(format.sigName[format.sigCount], ".");
        }
        strcat(format.sigName[format.sigCount], wfDef->wfName);
    } else
        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)(size_t)index;
}

/* Define the flush function */
int flushSAF(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 )
        {
            char* lStr = getLogicStr((int)sig.val);
            fprintf((FILE*) stream, "\t%s \t%e  \t%s\n",name, sig.time, lStr );
        }
        else if ( sig.isComplex )
            fprintf((FILE*) stream, "\t%s \t%e  \t%e  \t%e\n", name, sig.time,
sig.val, sig.imag );
        else
            fprintf((FILE*) stream, "\t%s \t%e  \t%e\n", name, sig.time, sig.val);
    }
    /*  Reset the counter */
```

```
    format.dataCount = 0;
    return 0;
}

/* Defines the close API */
int closeSAF(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 addDWfPointSAF(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 */
    int index = (size_t)(wfHandle);

    if( format.dataCount < MAX_DATA_COUNT )
    {
        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;
    }
    else
        printf("Exceeded the maximum data point that can be stored \n");

    return 0;
}

/* Defines addAWfPoint API */
int addAWfPointSAF(uwi_StreamHandle stream, uwi_WfHandle wfHandle, double val,
                                                        double t)
{
```

```
    int index = (size_t)(wfHandle);


    if( format.dataCount < MAX_DATA_COUNT )
    {
        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;
    }
    else
        printf("Exceeded the maximum data point that can be stored \n");
    return 0;
}

/* Defines addAWfComplexPoint API */
int addAWfComplexPointSAF(uwi_StreamHandle stream, uwi_WfHandle wfHandle,
                          double vReal, double vImag, double freq)
{
    int index = (size_t)(wfHandle);


    if( format.dataCount < MAX_DATA_COUNT )
    {
        format.signal[format.dataCount].time      = freq;
        format.signal[format.dataCount].val       = vReal;
        format.signal[format.dataCount].imag      = vImag;
        format.signal[format.dataCount].index     = index;
        format.signal[format.dataCount].isDigital = 0;
        format.signal[format.dataCount].isComplex = 1;
        ++format.dataCount;
    }
    else
        printf("Exceeded the maximum data point that can be stored \n");
    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        = openSAF;
    wfIntDef.defineWf    = defineWfSAF;
    wfIntDef.addDWfPoint = addDWfPointSAF;
    wfIntDef.addAWfPoint = addAWfPointSAF;
    wfIntDef.addAWfComplexPoint = addAWfComplexPointSAF;
    wfIntDef.flush       = flushSAF;
    wfIntDef.close       = closeSAF;
    wfIntDef.getErrMsg   = NULL;
    wfIntDef.resetXCoord = NULL;

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