ICS 31.200
CCS L 70

EDA开放创新合作机制团体标准

T/EDA² 00X—2023

数字波形接口

Digital Waveform Application Programmers Interface

(征求意见稿)

2023-XX-XX 发布　　2023-XX-XX 实施

EDA 开放创新合作机制（EDA²） 发 布

版权保护文件

目录

前言

本文件按照 GB/T 1.1—2020《标准化工作导则第 1 部分：标准化文件的结构和起草规则》的规则起草。

请注意本文件的某些内容可能涉及专利。本文件的发布机构不承担识别这些专利的责任。

本部分由 EDA 开放创新合作机制（EDA²）提出并归口。

本标准的主要起草单位：复旦大学、宁波大学、浙江大学、芯华章科技股份有限公司、上海阿卡思微电子技术有限公司、上海概伦电子股份有限公司、北京华大九天科技股份有限公司、上海合见工业软件集团有限公司、深圳海思半导体有限公司、上海国微思尔芯技术股份有限公司。

本标准的主要起草人：杨帆、储著飞、卓成、尹勋钊、黄世杰、郑艺斌、李义辉、汪燕芳、田野、郭根华、高波、吴倩瑜、林铠鹏、王锐、齐正华、范志英、王宗源、王祥凯。

数字波形接口

1. 范围

本文件制定的主要范围为波形生产数据工具和调试工具与波形数据交互接口的技术要求，即对不同类型的波形数据进行规范化的存储及读取操作。

本标准适用于指导芯片波形产生工具和调试工具与波形数据接口之间的开发及使用。

2. 规范性引用文件

IEEE Std 1800™-2017 系统 verilog 标准——统一的硬件设计、规范和验证语言（IEEE Standard for SystemVerilog--Unified Hardware Design, Specification, and Verification Language）。

3. 术语和定义

下列术语和定义适用于本文件。

3.1.

数字波形 digital waveform

数字信号产生的二进制数据在时间上分布情况的图形抽象。

3.2.

数字波形底座 digital wave form database

数字波形存储到物理媒介时所形成的文件。

4. 缩略语

下列缩略语适用于本文件:

API 应用程序接口（Application Programming Interface）

DB 数据库(DataBase)

VCD 值变转储 （value change dump）

5. 概述

当进行数字芯片仿真时，包括前端仿真工具等在内的芯片验证工具应通过数字波形底座提供的写接口进行波形数据导出；包括调试工具等在内的芯片验证工具应通过依赖数字波形底座提供的读接口，进行波形数据的读取。数字波形底座应提供如下功能接口：

a) 写接口：用于接收来自芯片验证工具的数字波形数据，并将其存储在本地；

b） 读接口：用于将本地保存的数字波形底座文件加载到芯片验证工具；
c） 维测接口：用于维护测试日志信息。

6. 通用定义

6.1. RetCode 状态码定义（枚举）

状态码	含义
RetCodeOk	成功
RetCodeError	失败

6.2. 日志类型 LogType 定义（枚举）

类型	含义
LogTypeDebug	打印DEBUG、INFO、WARNING、ERROR维测
LogTypeInfo	打印INFO、WARNING、ERROR维测
LogTypeWarning	打印WARNING、ERROR维测
LogTypeError	打印ERROR维测

6.3. 数据库操作码 DbOptionCode 定义（枚举）

类型	含义
DbOptionRead	读取数据库所有内容
DbOptionWrite	写模式打开数据库
DbOPtionRemoteRead	远程读模式打开数据库
DbOPtionRemoteWrite	远程写模式打开数据库

6.4. VarType 定义（枚举）

类型	含义
VarTypeNone	Invalid 类型
VarTypeWire	Wire
VarTypeBit	Bit
VarTypeReg	Reg
VarTypeShortInt	ShortInt
VarTypeInt	Int
VarTypeInteger	Integer
VarTypeLongInt	LongInt
VarTypeByte	Byte
VarTypeShortReal	ShortReal
VarTypeReal	Real
VarTypeTime	Time
VarTypeString	String

VarTypeEvent	Event
VarTypeParameter	Parameter
VarTypeEnum	Enum（对应数据为100）
VarTypeStruct	Struct
VarTypeUnion	Union
VarTypePackedStruct	PackedStruct
VarTypePackedUnion	PackedUnion
VarTypePackedArray	PackedArray
VarTypeArray	Array

6.5. ScopeType 定义（枚举）

类型	含义
ScopeTypeNone	Invalid 类型
ScopeTypeModule	Module
ScopeTypeTask	Task
ScopeTypeFunc	Function
ScopeTypeBlock,	Block
ScopeTypePackage	Package
ScopeTypeProgram	Program
ScopeTypeGenerate	Generate
ScopeTypeInterface	Interface
ScopeTypeModPort	Port
ScopeTypeClocking	Clock

6.6. Object 类型

类型	含义
ObjectTypeVar	Var
ObjectTypeScope	Scope

6.7. VarPortDirection 定义（枚举）

类型	含义
VarPortDirectionInput	Input
VarPortDirectionOutput	Output
VarPortDirectionInOut	In/output

6.8. DbPropertyType 定义（枚举）

类型	含义
DbPropTypeNone,	Invalid 类型
DbPropTypeVersion	Db版本

DbPropTypeCreater	创建时间

6.9. Property type 定义（枚举）

类型	含义
PropertyTypeNone	Invalid 类型
PropertyTypeName	Name
PropertyTypeObjType	返回值为ObjectType类型，var or scope
PropertyTypeType	当为var时，返回对应VarType，当为scope时，返回ScopeType
PropertyTypeBits	Bits
PropertyTypeArrayDim	维度信息
PropertyTypePackedArrayDim	PackedArrayDim
PropertyTypePortDir	PortDirection
PropertyTypeAggrMemberNum	Aggr var的成员数量
PropertyTypeSubVarNum	Scope的var数量
PropertyTypeSubScopeNum	Scope的子scope数量

6.10. Time scale 定义（枚举）

类型	含义
TimeScaleTypeNone	Invalid 类型
TimeScale1s	1s,
TimeScale100ms	100ms,
TimeScale10ms	10ms,
TimeScale1ms	1ms,
TimeScale100us	100us,
TimeScale10us	10us,
TimeScale1us	1us,
TimeScale100ns	100ns,
TimeScale10ns	10ns,
TimeScale1ns	1ns,
TimeScale100ps	100ps,
TimeScale10ps	10ps,
TimeScale1ps	1ps,
TimeScale100fs	100fs,
TimeScale10fs	10fs,
TimeScale1fs	1fs

6.11. 信号状态 SignalStatus 定义（枚举）

状态	含义

SignalStatus0	‘0’，VCD标准0态
SignalStatus1	‘1’，VCD标准1态
SignalStatusZ	‘z’，VCD标准z态
SignalStatusX,	‘x’，VCD标准x态

6.12. Value type 定义（枚举）

类型	含义
ValueTypeString	String
ValueTypeBitStr	BitStr
ValueTypeEnumStr	EnumStr
ValueTypeScalar	Scalar
ValueTypeReal	Real
ValueTypeInt	Int
ValueTypeUint	Uint
ValueTypeInt64	Int64
ValueTypeUint64	Uint64
ValueTypeVector	Vector
ValueTypeComplex	复数
ValueTypeComposite	Struct类型

6.13. 回调函数类型定义

1） 错误码回调函数

typedef void(*RetCodeCallBack)(RetCode code, void* userParam);

2） 维测回调函数指针定义

typedef void(*PrintLogCallBack)(LogType level, const char* moduleName, const char* log);

3） Hierarchy 遍历回调函数类型

typedef void(*HierarchyCallback)(ObjHandle h, void* userParam);

4） 遍历 var change 值回调函数类型

typedef void(*VcCallback)(VcTime time, VarHandle var, VarValue* value, void* userParam);

5） Var 时间变换回调函数类型

typedef void(*VcTimeCallback)(VcTime time, void* userParam)。

6.14. 基础数据结构定义

类型	含义
DbHandle(void *)	数据库句柄
ObjHandle(void *)	Var/Scope句柄
VarHandle(void *)	Var句柄
ScopeHandle(void *)	Scope句柄

VcIterHandle(void *)	Vc迭代器
VarTypeHandle(void *)	Var数据类型句柄
VcTime(uint64_t);	Vc时间戳
VarValuePtr (VarValue*)	varValue指针
UId(uint64_t)	Var的id

6.15. 时间区间结构定义

```
typedef struct TimeRange {
    uint64_t begin;
    uint64_t end;
} TimeRange;
```

说明：时间约定为前闭后开，[begin, end)。

6.16. var 数据类型结构定义

```
typedef struct VarTypeInfo {
    VarType varType;
    VarTypeHandle typeHandle; //当varType为基础数据类型时，typeHandle=NULL;
}VarTypeInfo;
```

1) 基本数据类型

```
typedef struct VarDataType {
    const char* memberName;
    VarTypeInfo varType;
} VarDataType;
```

2) Array 类型

```
typedef struct ArrayVar {
    uint8_t num;
    int *dimData[2];  //维度信息
    VarTypeInfo varType;
} ArrayVar;
```

3) 枚举类型

```
typedef struct EnumVar {
    int num;
    VarDataType varType;
    VarValuePtr values;
} EnumVar;
```

4) 组合类型

```
typedef struct CompositeVar{
    int num;
    VarDataType* vars;
} AggrVar;
```

5) 向量类型

```
typedef struct VectorValue {
    int num; // bit位数，实际的values的size为num / 32 + num %32 == 0 ? 0: 1;
    struct {
        uint32_t aVal, bVal;
    }* values;
} VectorValue;
```

6) 组合类型值类型

```
typedef struct CompositeVarValue {
    int num;    // 成员数量
    VarValuePtr values;
} CompositeVarValue;
```

6.17. 信号数据结构定义

```
typedef struct VarValue {
    ValueType valueType;
    union {
        const char* strValue;
        SignalStatus scalarValue;
        double realValue;
        double complexValue[2];
        int32_t int32Value;
        uint32_t uint32Value;
        int64_t int64Value;
        uint64_t uint64Value;
        VectorValue vectorValue;
        CompositeVarValue compositeValue;
    } value;
} VarValue;
```

6.18. sweep 属性结构定义

```
typedef struct {
    int32_t count;      // sweep count
    int32_t varCnt;     // variable count
    char** varNames;    // variable names
} SweepAttr;
```

7. 公共功能接口

7.1. 返回码回调函数绑定

函数原型	void set_cb_retcode(RetCode code, RetCodeCallBack cbFunc, void* userParam);	
函数功能	为返回码绑定回调函数，支持绑定多个回调函数	
调用约束		
参数	参数名	含义
	Code	RetCode返回码
	CbFunc	回调函数
	UserParam	用户参数
返回值	Void	

7.2. 获取错误码描述

函数原型	const char* get_error_msg(RetCode code);	
函数功能	获取错误码描述	
调用约束		
参数	参数名	含义
	Code	RetCode返回码
返回值	Const char*	错误码描述 未定义时，返回NULL

7.3. 日志回调函数绑定

函数原型	void regist_cb_log(PrintLogCallBack cbFunc, LogType level, const char* modelName);	
函数功能	日志回调函数绑定，当未绑定回调函数时，不进行日志打印	
调用约束		
参数	参数名	含义
	CbFunc	回调函数
	level	日志级别，类型为LogType
	modelName	模块名称
返回值	Void	

8. 读功能接口

8.1. 数据库管理

8.1.1. 打开 DB

函数原型	DbHandle open_db_read(const char *name, DbOptionCode options);	
函数功能	打开数据库，返回数据库句柄	
调用约束		
参数	参数名	含义
	Name	数据库路径信息

	Options	操作选项
返回值	DbHandle	执行成功返回句柄，否则返回NULL

8.1.2. 关闭 DB

函数原型	void close_db_read(DbHandle db)	
函数功能	关闭数据库	
调用约束		
参数	参数名	含义
	db	数据库句柄
返回值	void	

8.2. DB 属性信息查询

8.2.1. 查询 db 属性值

函数原型	const char* get_db_property(DbHandle db, DbPropertyType dbPropType);	
函数功能	获取db属性信息	
调用约束	数据库正常打开	
参数	参数名	含义
	db	数据库句柄
	dbPropertype	db属性类型(枚举类型：DbPropertyType)
返回值	const char*	当不存在的属性时，返回NULL;

8.2.2. 查询 TimeScale

函数原型	TimeScale get_time_scale(DbHandle db)	
函数功能	获取时间精度信息	
调用约束	数据库正常打开	
参数	参数名	含义
	db	数据库句柄
返回值	TimeScale	枚举TimeScale数据

8.2.3. 查询仿真起始时间

函数原型	VcTime get_begin_time(DbHandle db);	
函数功能	获取仿真起始时间	
调用约束	数据库正常打开	
参数	参数名	含义
	db	数据库句柄

返回值	VcTime	起始时间信息

8.2.4. 查询仿真结束时间

函数原型	VcTime get_end_time(DbHandle db);	
函数功能	获取仿真结束时间	
调用约束	数据库正常打开	
参数	参数名	含义
	db	数据库句柄
返回值	VcTime	结束时间信息

8.2.5. 查询信号 glitch 值

函数原型	Unsigned int get_max_glitch(DbHandle db);	
函数功能	获取glitches	
调用约束	数据库正常打开	
参数	参数名	含义
	db	数据库句柄
返回值	Unsigned int	

8.3. Hierarchy 数据查询

8.3.1. 遍历 hierarchy

8.3.1.1. 遍历顶层 module，仅遍历一层

函数原型	void traverse_db_top(DbHandle db, HierarchyCallback func, void* userParam);	
函数功能	遍历顶层hierarchy数据，仅遍历一层	
调用约束		
参数	参数名	含义
	db	数据库句柄
	func	遍历过程中的回调函数
	userParam	回调函数用户自定义参数
返回值	Void	

8.3.1.2. 遍历指定 scope 的子 scope，仅遍历一层

函数原型	void traverse_db_scopes(ScopeHandle h, HierarchyCallback func, void *param);	
函数功能	遍历scope h的子scope，仅遍历一层	
调用约束		
参数	参数名	含义
	h	待遍历的scope句柄
	Func	遍历过程中的回调函数
	param	回调函数用户自定义参数
返回值	Void	

8.3.1.3. 遍历指定 scope 的 var，仅遍历一层

函数原型	void traverse_db_vars(ScopeHandle h, HierarchyCallback func, void *param);	
函数功能	遍历scope h的所有vars，仅遍历一层	
调用约束		
参数	参数名	含义
	h	待遍历的scope句柄
	Func	遍历过程中的回调函数
	param	回调函数用户自定义参数
返回值	Void	

8.3.2. 获取 hierarchy 属性信息

8.3.2.1. 获取 string 类型属性

函数原型	const char* get_property_str(ObjHandle h, PropertyType prop)	
函数功能	获取句柄h的属性类型为prop的属性值类型为string的属性值	
调用约束		
参数	参数名	含义
	h	Var/scope句柄
	Prop	枚举类型属性值
返回值	Const char *	成功：返回const char* 失败：返回NULL

8.3.2.2. 获取 int 类型属性

函数原型	int get_property_int (ObjHandle h, PropertyType prop)	
函数功能	获取句柄h的属性类型为prop的属性值类型为int的属性值	
调用约束		
参数	参数名	含义
	h	Var/scope句柄
	Prop	枚举类型属性值
返回值	Int	

8.3.2.3. 获取 var 的 dim 信息

函数原型	int get_var_dims(VarHandle var, PropertyType prop, int (*buf)[2], uint32_t size);	
函数功能	获取信息句柄为var的维度信息，返回具体维度数值	
调用约束	Var为VarTypePackedArray或VarTypeArray类型的信号	
参数	参数名	含义
	h	Var/scope句柄
	porp	维度类型：取值为PropertyTypeArrayDim或 PropertyTypePackedArrayDim,
	buf	缓存，用于存放查询结果

	size	Buf长度
返回值	int	实际获取的维度数值

8.3.3. 获取父 scope

函数原型	ScopeHandle get_parent_handle(ObjHandle h);	
函数功能	获取scope/var对象的父scope	
调用约束		
参数	参数名	含义
	h	子scope/var句柄
返回值	ScopeHandle	返回父scope/var句柄 当h为顶层scope时，返回nullptr

8.3.4. 根据全名获取 scope/var 句柄

函数原型	ObjHandle get_handle_by_fullname(DbHandle db, const char* name);	
函数功能	根据scope/var命名获取对应的句柄	
调用约束		
参数	参数名	含义
	db	数据库句柄
	Name	Scope/var全名
返回值	ObjHandle	成功：返回scope/var句柄 失败：返回nullptr

8.3.5. 根据名称获取 scope/var 句柄

函数原型	ObjHandle get_handle_by_name (const char* name, ScopeHandle scope);	
函数功能	根据scope/var名称获取指定scope内对象的句柄	
调用约束		
参数	参数名	含义
	Name	Scope/var全名
	Scope	scope句柄
返回值	ObjHandle	成功：返回scope/var句柄 失败：返回nullptr

8.3.6. 根据名称获取 aggr 类 var 的成员句柄

函数原型	VarHandle get_composite_member_by_name (const char* name, VarHandle handle);	
函数功能	根据成员名称获取指定aggr类型的var（Struct/Union）的成员对象的句柄	
调用约束		
参数	参数名	含义
	Name	成员名称
	handle	Aggr类var对象句柄
返回值	VarHandle	成功：返回var句柄

		失败：返回nullptr

8.3.7. 获取 aggr 类 var 的成员句柄

函数原型	Int32_t get_composite_member_objects (VarHandle h, VarHandle* buf, uint32_t sizeBuf);	
函数功能	获取aggr类var h的所有成员句柄，保存到buf中	
调用约束	H为aggr类var（struct或Union）	
参数	参数名	含义
	h	Var句柄
	Buf	存入成员句柄
	sizeBuf	Buf大小
返回值	Int32_t	成功：实际获取的成员数量 失败：-1

8.3.8. 获取 enum 类 var 的定义

函数原型	RetCode get_enum_def (VarHandle var, EnumVar* value);	
函数功能	获取enum类var h的定义	
调用约束	H为enum类var	
参数	参数名	含义
	var	Var句柄
	Value	存入var的定义
返回值	RetCode	成功：RetCodeOk 失败：RetCodeError

8.3.9. 释放枚举类型结构

函数原型	void free_enum_def(EnumVar* value);	
函数功能	释放value内存	
调用约束		
参数	参数名	含义
	Value	EnumVar类型指针
返回值	Void	

8.3.10. 释放 scope/var 句柄

函数原型	void free_handle(ObjHandle h);	
函数功能	释放scope、var句柄，系统会自动释放，非必需执行项	
调用约束		
参数	参数名	含义
	h	待释放的Scope/var句柄
返回值	Void	

8.3.11. 获取 dumpon/dumpoff 数据

函数原型	int get_dumpoff_info(DbHandle db, TimeRange *buf, uint32_t sizeBuf,);	
函数功能	获取dumpon/dumpoff数据，保存到buf中	
调用约束		
参数	参数名	含义
	db	数据库句柄
	Buf	用于存放查询结果数据
	sizeBuf	Buf大小
返回值	Int	成功：返回实际获取的dumponoff数量 失败：返回-1

8.3.12. 释放 VarType 资源

函数原型	void free_var_type(VarTypeHandle h);	
函数功能	释放VarType，非必需执行	
调用约束		
参数	参数名	含义
	h	VarType句柄
返回值	Void	

8.3.13. 获取 scope/var id

函数原型	UId get_obj_uid_read(ObjHandle h)	
函数功能	根据var/scope句柄获取id	
调用约束		
参数	参数名	含义
	h	var/scope句柄
返回值	UId	var/scope的id

8.3.14. 获取 scope/var handle

函数原型	ObjHandle get_obj_handle_by_id_read(DbHandle h, UId objId);	
函数功能	根据var/scope获得对应的handle	
调用约束		
参数	参数名	含义
	h	数据库句柄
	objId	Scope/var id
返回值	ObjHandle	成功：scope/var句柄 失败：nullptr

8.3.15. 通过下标获取 Array 成员句柄

函数原型	VarHandle get_array_member_by_index_read(VarHandle arrayVar, uint8_t dims, const

	int* indexes[2])	
函数功能	根据var/scope获得对应的handle	
调用约束	arrayVar为array类型的var	
参数	参数名	含义
	ArrayVar	ArrayVar句柄
	Dims	维度
	Indexes	维度下标信息
返回值	VarHandle	成功： var句柄 失败：nullptr

8.4. 波形数据管理

8.4.1. 仿真数据加载与卸载

8.4.1.1. 仿真数据加载

函数原型	RetCode load_vc(DbHandle db);	
函数功能	加载var列表中，通过set_vc_range设置了range范围的仿真数据	
调用约束		
参数	参数名	含义
	db	数据库句柄
返回值	RetCode	成功：RetCodeOk 失败：RetCodeError

8.4.1.2. 通过 var 句柄卸载仿真数据

函数原型	void unload_vc_by_handle(VarHandle sig);	
函数功能	卸载通过load_vc加载的仿真数据	
调用约束		
参数	参数名	含义
	sig	vc句柄
返回值	Void	

8.4.1.3. 根据 UID 卸载指定信号数据

函数原型	void unload_vc_by_uid(DbHandle db, UId sig);	
函数功能	卸载已经加载的信号sig的仿真数据	
调用约束		
参数	参数名	含义
	db	数据库句柄
	Sig	信号id
返回值	Void	

8.4.2. Vc range 管理

8.4.2.1. Vc range 设置(回调查询)

函数原型	void set_cb_range(DbHandle db, VcTime startTime, VcTime endTime);	
函数功能	设置需要查询的vc range范围	
调用约束	时间区间包含startTime与endTime	
参数	参数名	含义
	db	枚举类型属性值
	startTime	起始时间
	endTime	结束时间
返回值	Void	

8.4.2.2. Vc range 重置（回调查询）

函数原型	void reset_cb_range(DbHandle db)	
函数功能	重置vc range为数据库全范围	
调用约束		
参数	参数名	含义
	db	数据库句柄
返回值	Void	

8.4.2.3. Vc range 设置(迭代器查询)

函数原型	void set_iter_range(DbHandle db, VcTime startTime, VcTime endTime);	
函数功能	设置需要查询的vc range范围	
调用约束	时间区间包含startTime与endTime	
参数	参数名	含义
	db	枚举类型属性值
	startTime	起始时间
	endTime	结束时间
返回值	Void	

8.4.2.4. Vc range 重置（迭代器查询）

函数原型	void reset_iter_range(DbHandle db)	
函数功能	重置vc range为数据库全范围	
调用约束		
参数	参数名	含义
	db	数据库句柄
返回值	Void	

8.4.3. 待遍历/查询 var 管理

8.4.3.1. 通过 var 句柄添加待查询 Var（回调方式查询）

函数原型	void add_cb_var_handle(VarHandle sig);

函数功能	设置需要查询的var句柄	
调用约束		
参数	参数名	含义
	sig	待添加的信号句柄
返回值	Void	

8.4.3.2. 通过 UID 待查询 Var 添加（回调方式查询）

函数原型	void add_cb_var_uid(DbHandle db, UId id);	
函数功能	设置需要查询的var句柄	
调用约束		
参数	参数名	含义
	db	待添加的信号句柄
	id	Var 的id
返回值	Void	

8.4.3.3. 待查询 Var 列表重置（回调方式查询）

函数原型	void reset_cb_vars(DbHandle db);	
函数功能	重置待查询的var列表，默认为无信号	
调用约束		
参数	参数名	含义
	db	数据库句柄
返回值	Void	

8.4.3.4. 通过 var 句柄添加待查询 Var（迭代器方式查询）

函数原型	void add_iter_var_handle(VarHandle sig);	
函数功能	设置需要查询的var句柄	
调用约束		
参数	参数名	含义
	sig	待添加的信号句柄
返回值	Void	

8.4.3.5. 通过 UID 待查询 Var 添加（迭代器方式查询）

函数原型	void add_iter_var_uid(DbHandle db, UId id);	
函数功能	设置需要查询的var句柄	
调用约束		
参数	参数名	含义
	db	待添加的信号句柄
	id	Var 的id
返回值	Void	

8.4.3.6. 待查询 Var 列表重置（迭代器方式查询）

函数原型	void reset_iter_vars(DbHandle db);	
函数功能	重置待查询的var列表，默认为无信号	
调用约束		
参数	参数名	含义
	db	数据库句柄
返回值	Void	

8.4.4. 遍历波形数据

函数原型	void traverse_all_vc(DbHandle db, VcCallback vcFunc, void* vcUserParam, VcTimeCallback timdFunc, void* timeUserParam);	
函数功能	遍历波形数据	
调用约束		
参数	参数名	含义
	db	数据库句柄
	vcFunc	VcCallback回调函数
	vcUserParam	VcCallback用户自定义参数
	timdFunc	VcTimeCallback回调函数
	timeUserParam	VcTimeCallback自定义参数
返回值	void	

8.4.5. Vc 迭代器管理

8.4.5.1. 普通 vc 迭代器获取

函数原型	VcIterHandle get_vc_iter_by_handle(ObjHandle sig);	
函数功能	获取信号sig的vc迭代器	
调用约束	数据通过load_vc正确加载	
参数	参数名	含义
	Sig	Var句柄
返回值	VcIterHandle	成功：返回vc迭代器 失败：返回nullptr

8.4.5.2. 多维 vc 迭代器获取

函数原型	VcIterHandle get_vc_iter_by_index(ObjHandle sig, int dims, const int *indexes[2]);	
函数功能	获取多维vc指定var的迭代器	
调用约束	数据通过load_vc正确加载	
参数	参数名	含义
	Sig	多维Var句柄
	Dims	维度信息

	Indexes	所关心的维度数据
返回值	VcIterHandle	成功：返回vc迭代器 失败：返回nullptr

8.4.5.3. Vc 迭代器销毁

函数原型	void free_iter(VcIterHandle h);	
函数功能	销毁vc迭代器对象，系统会自动执行，非必需执行项	
调用约束		
参数	参数名	含义
	h	Vc迭代器
返回值	Void	

8.4.5.4. 获取迭代器 Vc 的最小时间

函数原型	RetCode vc_get_min_time(VcIterHandle h, VcTime *time);	
函数功能	获取迭代器vc的最小时间值	
调用约束		
参数	参数名	含义
	h	Vc迭代器
	time	用于存放查询结果
返回值	RetCode	成功：RetCodeOk 失败：RetCodeError

8.4.5.5. 获取迭代器 Vc 的最大时间

函数原型	RetCode vc_get_max_time(VcIterHandle h, VcTime *time);	
函数功能	获取迭代器vc的最大时间值	
调用约束		
参数	参数名	含义
	h	Vc迭代器
	time	用于存放查询结果
返回值	RetCode	成功：RetCodeOk 失败：RetCodeError

8.4.5.6. 迭代器 Vc 设置为 begin

函数原型	RetCode vc_goto_start(VcIterHandle h);	
函数功能	迭代器vc设置为begin	
调用约束		
参数	参数名	含义
	h	Vc迭代器
返回值	RetCode	成功：RetCodeOk 失败：RetCodeError

8.4.5.7. 迭代器 Vc 指向指定时间戳

函数原型	RetCode vc_goto_time(VcIterHandle h, VcTime time);	
函数功能	迭代器vc指向指定时间戳	
调用约束		
参数	参数名	含义
	h	Vc迭代器
	time	时间戳
返回值	RetCode	成功：RetCodeOk 失败：RetCodeError

8.4.5.8. 迭代器 Vc ++

函数原型	RetCode vc_goto_next(VcIterHandle h)	
函数功能	迭代器vc++	
调用约束		
参数	参数名	含义
	h	Vc迭代器
返回值	RetCode	成功：RetCodeOk 失败：RetCodeError

8.4.5.9. 迭代器 Vc--

函数原型	RetCode vc_goto_prev(VcIterHandle h);	
函数功能	迭代器vc--	
调用约束		
参数	参数名	含义
	h	Vc迭代器
返回值	RetCode	成功：RetCodeOk 失败：RetCodeError

8.4.5.10. 获取迭代器 Vc 到指定的时间戳

函数原型	VcTime vc_get_time(VcIterHandle h);	
函数功能	设置迭代器到指定时间戳	
调用约束		
参数	参数名	含义
	h	Vc迭代器
	time	用于存放查询
返回值	VcTime	迭代器当前的时间

8.4.5.11. 获取迭代器 Vc 对应的信号数据

函数原型	RetCode vc_get_value_ex(VcIterHandle h, VarValue *value);

函数功能	获取迭代器vc对应的信号数据，vc_get_value接口的扩展，获取结构化的value数据	
调用约束		
参数	参数名	含义
	h	Vc迭代器
	Value	用于存放信号数据
返回值	RetCode	成功：RetCodeOk 失败：RetCodeError

8.4.6. 其它

8.4.6.1. Db 数据刷新

函数原型	void sync_db(DbHandle dbhandle)	
函数功能	同步db数据，在同时读写场景会用到	
调用约束		
参数	参数名	含义
	dbhandle	Db句柄
返回值	Void	

8.4.7. 模拟特有接口

8.4.7.1. 获取 Sweep 属性

函数原型	SweepAttr* get_sweep_attribution(DbHandle db);	
函数功能	获取sweep属性信息	
调用约束		
参数	参数名	含义
	db	Db句柄
返回值	Double*	失败：nullptr

8.4.7.2. 销毁 Sweep 属性

函数原型	Void destroy_sweep_attribution(SweepAttr* attr);	
函数功能	销毁sweep属性信息	
调用约束		
参数	参数名	含义
	attr	SweepAttr属性
返回值	Void	

8.4.7.3. 销毁 Sweep 属性

函数原型	Void destroy_sweep_attribution(SweepAttr* attr);	
函数功能	销毁sweep属性信息	
调用约束		
参数	参数名	含义

	attr	SweepAttr属性
返回值	Void	

8.4.7.4. 获取 Sweep 属性值

函数原型	RetCode get_sweep_var_values(DbHandle db, int index, double* values);	
函数功能	获取sweep属性值	
调用约束		
参数	参数名	含义
	Db	Db句柄
	index	Sweep索引
	Values	用于存放sweep值
返回值	Void	

8.4.7.5. 获取同步模式

函数原型	bool get_synchronous_mode(DbHandle db);	
函数功能	同步异步标志读取	
调用约束		
参数	参数名	含义
	db	Db句柄
返回值	bool	true为同步模式; false为异步模式

9. 写功能接口

9.1. 数据库管理

9.1.1. 打开 DB

函数原型	DbHandle open_db_write(const char *name, DbOptionCode options);	
函数功能	打开数据库，返回数据库句柄	
调用约束		
参数	参数名	含义
	Name	数据库路径信息
	Options	DbOptionCode类DB操作选项
返回值	DbHandle	执行成功返回句柄，否则返回nullptr

9.1.2. 关闭 DB

函数原型	void close_db_write(DbHandle db)	
函数功能	关闭数据库	
调用约束		
参数	参数名	含义

	db	数据库句柄
返回值	void	

9.2. DB 属性信息查询

9.2.1. 设置 TimeScale

函数原型	void set_time_scale(DbHandle db, TimeScale ts);	
函数功能	设置时间精度信息	
调用约束	数据库正常打开	
参数	参数名	含义
	Db	数据库句柄
	Ts	时间精度信息
返回值	Void	

9.2.2. 写入仿真时间

函数原型	void set_vc_time(DbHandle db, VcTime time);	
函数功能	写入仿真时间	
调用约束	数据库正常打开	
参数	参数名	含义
	db	数据库句柄
	Time	仿真时间
返回值	Void	

9.2.3. 设置波形数据的 glitches

函数原型	void set_max_glitch(DbHandle db, unsigned int glitches);	
函数功能	设置波形数据的glitches	
调用约束	数据库正常打开	
参数	参数名	含义
	db	数据库句柄
	glitches	Glitches值
返回值		

9.2.4. 设置 db 属性值

函数原型	void set_db_property(DbHandle db, DbPropertyType dbPropType, const char* value);	
函数功能	设置db属性信息	
调用约束	数据库正常打开，value长度不超过256B	
参数	参数名	含义
	db	数据库句柄
	dbPropType	Db属性类型
	Const char*	属性值

返回值	void	

9.3. 写入 Dumpon/dumpoff

9.3.1. 写入 dumpon

函数原型	void set_dumpon(DbHandle db);	
函数功能	写入dumpon	
调用约束	数据库正常打开	
参数	参数名	含义
	Db	数据库句柄
返回值	Void	

9.3.2. 写入 dumpoff

函数原型	void set_dumpoff(DbHandle db);	
函数功能	写入dumpoff	
调用约束	数据库正常打开	
参数	参数名	含义
	Db	数据库句柄
返回值	Void	

9.4. 写入 Hierarchy 数据

9.4.1. 创建 scope

函数原型	ScopeHandle create_scope(DbHandle db, const char *name, ScopeType type, const char* moduleName = nullptr);	
函数功能	创建scope	
调用约束	数据库正常打开	
参数	参数名	含义
	Db	数据库句柄
	name	Scope名称，支持全名创建scope
	Type	Scope类型
	moduleName	Module名称，默认为nullptr
返回值	ScopeHandle	成功：返回scope句柄 失败：返回nullptr

9.4.2. 创建 upscope

函数原型	void create_upscope(DbHandle db);	
函数功能	创建upscope	
调用约束	数据库正常打开	
参数	参数名	含义
	Db	数据库句柄

返回值	void	

9.4.3. 创建 var 类型

9.4.3.1. 创建 packed 的数组类型的 var 类型

函数原型	VarTypeHandle create_packed_array_type(DbHandle db, ArrayVar *info);	
函数功能	创建packed的数组类型的var类型	
调用约束	数据库正常打开	
参数	参数名	含义
	Db	数据库句柄
	Info	packed的数组类型的var类型
返回值	VarTypeHandle	成功：返回var类型句柄 失败：返回nullptr

9.4.3.2. 创建 unpacked 的数组类型的 var 类型

函数原型	VarTypeHandle create_array_type(DbHandle db, ArrayVar *info);	
函数功能	创建unpacked的数组类型的var类型	
调用约束	数据库正常打开	
参数	参数名	含义
	Db	数据库句柄
	Info	packed的数组类型的var类型
返回值	VarTypeHandle	成功：返回var类型句柄 失败：返回nullptr

9.4.3.3. 创建 struct/union 类型的 var 类型

函数原型	VarTypeHandle create_composite_type (DbHandle db, CompositeVar *info, VarType ty);	
函数功能	创建struct/union类型的var类型	
调用约束	数据库正常打开,ty为VarTypeUnion或VarTypeStruct	
参数	参数名	含义
	Db	数据库句柄
	Info	枚举类型的var类型
	Type	struct/union类型标识
返回值	VarTypeHandle	成功：返回var类型句柄 失败：返回nullptr

9.4.3.4. 创建枚举类型的 var 类型

函数原型	VarTypeHandle create_enum_type(DbHandle db, EnumVar *info);	
函数功能	创建枚举类型的var类型	
调用约束	数据库正常打开	

参数	参数名	含义
	Db	数据库句柄
	Info	枚举类型的var类型
返回值	VarTypeHandle	成功：返回var类型句柄 失败：返回nullptr

9.4.4. 创建 var

9.4.4.1. 创建 var

函数原型	VarHandle create_var(DbHandle db, const char *name, VarTypeInfo* type, int dims[2]);	
函数功能	创建var	
调用约束	数据库正常打开	
参数	参数名	含义
	Db	数据库句柄
	Name	Var名称
	Type	Var类型
	dims	维度信息
返回值	VarHandle	成功：返回var句柄 失败：返回nullptr

9.4.4.2. 使用 id 创建 var

函数原型	VarHandle create_var_by_id(DbHandle db, const char *name, UId id);	
函数功能	使用var id创建新的var对象	
调用约束	数据库正常打开	
参数	参数名	含义
	Db	数据库句柄
	Name	名称
	Id	Var id
返回值	VarHandle	成功：返回var句柄 失败：返回nullptr

9.4.4.3. 使用 var 句柄创建 var

函数原型	VarHandle create_var_by_handle(DbHandle db, const char *name, VarHandle handle);	
函数功能	使用handle创建新的var对象，新创建的var与handle的id相同	
调用约束	数据库正常打开	
参数	参数名	含义
	Db	数据库句柄
	Name	名称
	Handle	Var句柄

返回值	VarHandle	成功：返回var句柄 失败：返回nullptr

9.4.4.4. 获取组合 var 的成员项

函数原型	VarHandle get_composite_member(VarHandle aggr, const char *name);	
函数功能	获取组合var的成员项的var对象	
调用约束	数据库正常打开	
参数	参数名	含义
	aggr	组合var
	Name	Var名称
返回值	VarHandle	成功：返回var句柄 失败：返回nullptr

9.4.4.5. 获取多维 Var 的成员项 var 句柄

函数原型	VarHandle get_array_member_by_index_write(VarHandle arrayVar, uint8_t dims, const int* indexes[2]);	
函数功能	获取多维Var的成员项var句柄	
调用约束	数据库正常打开	
参数	参数名	含义
	arrayVar	组合var句柄
	Dims	var维度
	Indexes	下标信息
返回值	VarHandle	成功：返回var句柄 失败：返回nullptr

9.4.5. 属性管理管理

9.4.5.1. Int 型属性设置

函数原型	void set_property_int(ObjHandle var, int prop, int val);	
函数功能	Var int型属性设置	
调用约束	数据库正常打开	
参数	参数名	含义
	Var	Var句柄
	Prop	属性标识
	Val	属性值
返回值	Void	

9.4.5.2. string 型属性设置

函数原型	void set_property_str(ObjHandle var, int prop, const char* val);
函数功能	Var string型属性设置

调用约束	数据库正常打开	
参数	参数名	含义
	Var	Var句柄
	Prop	属性标识
	Val	属性值
返回值	Void	

9.4.5.3. 根据 var 句柄获取 var 的 UID

函数原型	UId get_var_uid_write (ObjHandle var);	
函数功能	根据var句柄获取var id	
调用约束	数据库正常打开	
参数	参数名	含义
	Var	Var句柄
返回值	UId	

9.5. 仿真数据管理

9.5.1. 通过句柄写入仿真数据

函数原型	void set_value_by_handle(DbHandle db, ObjHandle var, VarValuePtr val);	
函数功能	通过句柄写入仿真数据	
调用约束	数据库正常打开	
参数	参数名	含义
	db	数据库句柄
	Var	Var句柄
	Val	信号值
返回值	Void	

9.5.2. 通过 id 写入仿真数据

函数原型	void set_value_by_id(DbHandle db, UId id, VarValuePtr val);	
函数功能	通过句柄及偏移量写入仿真数据，针对unpacked array	
调用约束	数据库正常打开	
参数	参数名	含义
	db	数据库句柄
	id	Var id
	Val	信号值
返回值	Void	

9.6. 其它

9.6.1. Db 数据主动落盘

函数原型	void flush_db(DbHandle dbhandle)

函数功能	Db数据主动落盘	
调用约束		
参数	参数名	含义
	dbhandle	Db句柄
返回值	Void	

9.7. 模拟特有接口

9.7.1. 创建 Sweep 属性对象

函数原型	RetCode create_sweep_attr (DbHandle db, const SweepAttr *sweepAttr);	
函数功能	创建sweep对象	
调用约束		
参数	参数名	含义
	db	Db句柄
	sweepAttr	属性对象指针
返回值	RetCode	成功：StateOK 失败：StateError

9.7.2. begin_sweep

函数原型	RetCode begin_sweep(DbHandle db, double *val);	
函数功能		
调用约束		
参数	参数名	含义
	db	Db句柄
	Val	Sweep属性值
返回值	RetCode	成功：RetCodeOk 失败：RetCodeError

9.7.3. EndSweep 值

函数原型	RetCode end_sweep (DbHandle db);	
函数功能	End sweep信息	
调用约束		
参数	参数名	含义
	db	Db句柄
返回值	RetCode	成功：RetCodeOk 失败：RetCodeError

9.7.4. 同步模式设置

函数原型	void set_synchronous_mode(DbHandle db,bool mode);
函数功能	设置同步异步标志

调用约束		
参数	参数名	含义
	db	Db句柄
	mode	bool=true为同步模式，bool=false为异步模式
返回值	void	

10. 波形文件转换接口

10.1. VCD 文件转换为数字波形底座

函数原型	RetCode generate_from_vcd(const char* vcd, const char* db)	
函数功能	将vcd文件转换为数字波形底座文件。	
调用约束		
参数	参数名	含义
	Vcd	Vcd文件名称（包含路径）
	db	Vsdb文件名称（包含路径）
返回值	RetCode	成功：RetCodeOk 失败：RetCodeError

10.2. 数字波形底座转换为 VCD

函数原型	RetCode convert_to_vcd(const char* dbName, const char* vcd);	
函数功能	将数字波形底座文件转换为vcd文件。	
调用约束		
参数	参数名	含义
	dbName	wavedb文件名称（包含路径）
	Vcd	Vcd文件名称（包含路径）
返回值	RetCode	成功：RetCodeOk 失败：RetCodeError

附录 A

数字波形底座与芯片验证工具之间的接口函数应基于符合本文件要求的数字波形接口头文件进行开发。数字波形接口头文件应符合如下要求：

```
/*
    wavedb api head file
*/

#ifndef WAVEDB_API_H
#define WAVEDB_API_H

#ifdef WIN32
#include <windows.h>
#else
#include <inttypes.h>
#include <unistd.h>
#endif

#ifdef __cplusplus
extern "C" {
#endif

typedef void* DbHandle;
typedef void* ObjHandle;
typedef void* ScopeHandle;
typedef void* VarHandle;
typedef void* VcIterHandle;
typedef void* VarTypeHandle;
typedef uint64_t VcTime;
typedef VarValue* VarValuePtr;
typedef uint64_t UId;

typedef enum RetCode {
    RetCodeOk,
    RetCodeError,
    // 各厂家可按需要扩展
} RetCode;

typedef enum LogType {
    LogTypeDebug,
    LogTypeInfo,
```

```
    LogTypeWarning,
    LogTypeError,
} LogType;

typedef enum DbOptionCode {
    DbOptionRead,
    DbOptionWrite,
    DbOPtionRemoteRead,
    DbOPtionRemoteWrite,
    // 各厂家可按需要扩展
} DbOptionCode;

typedef enum VarType {
    VarTypeNone,
    VarTypeWire,
    VarTypeBit,
    VarTypeReg,
    VarTypeShortInt,
    VarTypeInt,
    VarTypeInteger,
    VarTypeLongInt,
    VarTypeByte,
    VarTypeShortReal,
    VarTypeReal,
    VarTypeTime,
    VarTypeString,
    VarTypeEvent,
    VarTypeParameter,

    VarTypeEnum = 100,
    VarTypeStruct,
    VarTypeUnion,
    VarTypePackedStruct,
    VarTypePackedUnion,
    VarTypePackedArray,
    VarTypeArray,
} VarType;

typedef enum ScopeType {
    ScopeTypeNone,
    ScopeTypeModule,
```

```
        ScopeTypeTask,
        ScopeTypeFunc,
        ScopeTypeBlock,
        ScopeTypePackage,
        ScopeTypeProgram,
        ScopeTypeGenerate,
        ScopeTypeInterface,
        ScopeTypeModPort,
        ScopeTypeClocking,
    } ScopeType;

    typedef enum ObjectType {
        ObjectTypeVar,
        ObjectTypeScope,
    } ObjectType;

    typedef enum VarPortDirection {
        VarPortDirectionInput,
        VarPortDirectionOutput,
        VarPortDirectionInOut,
    } VarPortDirection;

    typedef enum DbPropertyType {
        DbPropNone,
        DbPropVersion,
        DbPropCreater
        // 各厂家可按需要扩展
    } DbPropertyType;

    typedef enum PropertyType {
        PropertyTypeNone,
        PropertyTypeName,
        PropertyTypeObjType,            // 用于查询ObjectHandle是scope还是var
        PropertyTypeType,               // 当ObjectHandle是scope,返回ScopeType;
否则，返回VarType
        PropertyTypeBits,
        PropertyTypeArrayDim,
        PropertyTypePackedArrayDim,
        PropertyTypePortDir,
        PropertyTypeCompositeMemberNum, // 组合类var中成员的数量
        PropertyTypeSubVarNum,
```

```
    PropertyTypeSubScopeNum,
} PropertyType;

typedef enum TimeScale {
    TimeScaleNone,
    TimeScale1s,
    TimeScale100ms,
    TimeScale10ms,
    TimeScale1ms,
    TimeScale100us,
    TimeScale10us,
    TimeScale1us,
    TimeScale100ns,
    TimeScale10ns,
    TimeScale1ns,
    TimeScale100ps,
    TimeScale10ps,
    TimeScale1ps,
    TimeScale100fs,
    TimeScale10fs,
    TimeScale1fs
} TimeScale;

typedef enum SignalStatus {
    SignalStatus0,
    SignalStatus1,
    SignalStatusZ,
    SignalStatusX,
} SignalStatus;

// 新增加，用于Struct类var value值数据结构
typedef struct AggrVarValue {
    int num;    // 成员数量
    VarValuePtr values;
} AggrVarValue;

typedef enum ValueType {
    ValueTypeString,
    ValueTypeBitStr,
    ValueTypeEnumStr,
    ValueTypeScalar,
```

```
        ValueTypeReal,
        ValueTypeInt,
        ValueTypeUint,
        ValueTypeInt64,
        ValueTypeUint64,
        ValueTypeVector,
        ValueTypeComplex,           // 复数
        ValueTypeAggr;        // 新增加Aggr类型，用于struct等类型，VarValue类型为CompositeVarValue
    } ValueType;

    // 时间约定为前闭后开，[begin, end)
    typedef struct TimeRange {
        uint64_t begin;
        uint64_t end;
    } TimeRange;

    // 基本数据类型
    typedef struct VarTypeInfo {
        VarType varType;
        VarTypeHandle typeHandle; //当varType为基础数据类型时，typeHandle=NULL;
    } VarTypeInfo;

    typedef struct VarDataType {
        const char* memberName;
        VarTypeInfo varType;
    } VarDataType;

    typedef struct ArrayVar {
        uint8_t num;
        int *dimData[2];
        VarTypeInfo varType;
    } ArrayVar;

    typedef struct EnumVar {
        int num;
        VarDataType varType;
        VarValuePtr values;
    } EnumVar;

    typedef struct CompositeVar {
```

```
    int num;
    VarDataType* vars;
} CompositeVar;

typedef struct VectorValue {
    int num; // bit位数，实际的values的size为num / 32 + num %32 == 0 ? 0: 1;
    struct {
        uint32_t aVal, bVal;
    }* values;
} VectorValue;

typedef struct VarValue {
    ValueType valueType;
    union {
        const char* strValue;
        SignalStatus scalarValue;
        double realValue;
        double complexValue[2];
        int32_t int32Value;
        uint32_t uint32Value;
        int64_t int64Value;
        uint64_t uint64Value;
        VectorValue vectorValue;
        CompositeVarValue compositeValue;
    } value;
} VarValue;

typedef struct {
    int32_t count;      // sweep count
    int32_t varCnt;     // variable count
    char** varNames;    // variable names
} SweepAttr;

#define WaveIsError(err) ((RetCodeOk) != 0)

typedef void(*RetCodeCallBack)(RetCode code, void* userParam);
typedef void(*PrintLogCallBack)(LogType level, const char* moduleName, const
char* log);
typedef void(*HierarchyCallback)(ObjHandle h, void* userParam);
typedef void(*VcCallback)(VcTime time, VarHandle var, VarValue* value, void*
userParam);
```

```
    typedef void(*VcTimeCallback)(VcTime time, void* userParam);

    // 为返回码设置回调函数
    void set_cb_retcode(RetCode code, RetCodeCallBack cbFunc, void* userParam);
    // 获取错误码
    const char* get_error_msg(RetCode code);
    void regist_cb_log(PrintLogCallBack cbFunc, LogType level, const char*
modelName);

    // WaveDB read api
    DbHandle open_db_read(const char *name, DbOptionCode options);
    void close_db_read(DbHandle db);
    const char* get_db_property(DbHandle db, DbPropertyType dbPropType);
    TimeScale get_time_scale(DbHandle db);
    VcTime get_begin_time(DbHandle db);
    VcTime get_end_time(DbHandle db);
    unsigned int get_max_glitch(DbHandle db);
    void traverse_db_top(DbHandle db, HierarchyCallback func, void* userParam);
    void traverse_db_scopes(ScopeHandle h, HierarchyCallback func, void
*userParam);
    void traverse_db_vars(ScopeHandle h, HierarchyCallback func, void *userParam);
    const char* get_property_str(ObjHandle h, PropertyType prop);
    int get_property_int(ObjHandle h, PropertyType prop);

    // 查询array dims/返回读取的维度数
    int get_var_dims(VarHandle var, PropertyType prop, int[][2] * buf, uint32_t
size);

    ObjHandle get_parent_handle(ObjHandle h);
    ObjHandle get_handle_by_fullname(DbHandle db, const char* name);
    // 获取scope的下一级var或scope句柄
    ObjHandle get_handle_by_name(ScopeHandle handle, const char* name);

    VarHandle get_composite_member_by_name(VarHandle h, const char* name);
    int32_t get_composite_member_objects(VarHandle h, VarHandle* buf, uint32_t
sizeBuf);
    // 获取枚举类型定义
    RetCode get_enum_def(VarHandle var, EnumVar* value);
    void free_enum_def(EnumVar* value);
    void free_handle(ObjHandle h);
    int get_dumpoff_info(DbHandle db,  TimeRange *buf, uint32_t sizeBuf);
```

```
    void free_var_type(VarTypeHandle h);

    UId get_obj_uid_read(ObjHandle h);
    ObjHandle get_obj_handle_by_id_read(DbHandle h, UId objId);
    VarHandle get_array_member_by_index_read(VarHandle arrayVar, uint8_t dims,
const int* indexes[2]);

    RetCode load_vc(DbHandle db);
    void unload_vc_by_handle(VarHandle sig);
    void unload_vc_by_uid(DbHandle db, UId sig);
    void set_cb_range(DbHandle db, VcTime startTime, VcTime endTime);
//[start,end]
    void reset_cb_range(DbHandle db);
    void add_cb_var_handle(VarHandle sig);
    void add_cb_var_uid(DbHandle db, UId id);
    void reset_cb_vars(DbHandle db);
    void traverse_all_vc(DbHandle db, VcCallback func, void* varUserParam,
VcTimeCallback func, void* timeUserParam);

    // 迭代器操作
    // 迭代器方式访问vc，设置需要加载数据信息，提升性能
    void set_iter_range(DbHandle db, VcTime startTime, VcTime endTime);
//[start,end]
    void reset_iter_range(DbHandle db);
    void add_iter_var_handle(VarHandle sig);
    void add_iter_var_uid(DbHandle db, UId id);
    void reset_iter_vars(DbHandle db);
    VcIterHandle get_vc_iter_by_handle(VarHandle sig);
    // 多维信号获取子信号的迭代器
    VcIterHandle get_vc_iter_by_index(VarHandle sig, int dims, const int
*indexes[2]);
    void free_iter(VcIterHandle h);
    RetCode vc_get_min_time(VcIterHandle h, VcTime *time);
    RetCode vc_get_max_time(VcIterHandle h, VcTime *time);
    RetCode vc_goto_start(VcIterHandle h);
    RetCode vc_goto_time(VcIterHandle h, VcTime time);
    RetCode vc_goto_next(VcIterHandle h);
    RetCode vc_goto_prev(VcIterHandle h);
    VcTime vc_get_time(VcIterHandle h);
    RetCode vc_get_value_ex(VcIterHandle h, VarValue *value);
    void sync_db(DbHandle dbhandle);
```

```
    // 模拟专有接口
    SweepAttr* get_sweep_attribution(DbHandle db);
    void destory_sweep_attr(SweepAttr* attr);
    RetCode get_sweep_var_values(DbHandle db, int index, double* values);
    bool get_synchronous_mode(DbHandle db);

    // WaveDB write api
    DbHandle open_db_write(const char *name, DbOptionCode options);
    void close_db_write(DbHandle db);
    void set_db_property(DbHandle db, int dbPropType, const char* value);
    void set_time_scale(DbHandle db, TimeScale ts);
    void set_vc_time(DbHandle db, VcTime time);
    void set_max_glitch(DbHandle db, unsigned int glitches);
    void set_dumpon(DbHandle db);
    void set_dumpoff(DbHandle db);

    // 创建与moduleName相同的scope，如不存在，则moduleName 为NULL
    ScopeHandle create_scope(DbHandle db, const char *name, ScopeType type, const
char* moduleName);
    void create_upscope(DbHandle db);
    VarTypeHandle create_packed_array_type(DbHandle db, ArrayVar *info);
    VarTypeHandle create_array_type(DbHandle db, ArrayVar *info);
    VarTypeHandle create_composite_type(DbHandle db, CompositeVar *info, VarType
type);
    VarTypeHandle create_enum_type(DbHandle db, EnumVar *info);
    VarHandle create_var(DbHandle db, const char *name, VarTypeInfo* type, int
dims[2]);
    VarHandle create_var_by_id(DbHandle db, const char *name, UId id);
    VarHandle create_var_by_handle(DbHandle db, const char *name, VarHandle
handle);
    VarHandle get_composite_member(VarHandle var, const char *name);
    VarHandle get_array_member_by_index_write(VarHandle arrayVar, uint8_t dims,
const int* indexes[2]);

    void set_property_int(ObjHandle var, PropertyType prop, int value);
    void set_property_str(ObjHandle var, PropertyType prop, const char* value);
    UId get_var_uid_write(ObjHandle var);
    void set_value_by_handle(DbHandle db, ObjHandle var, VarValuePtr value);
    void set_value_by_id(DbHandle db, UId id, VarValuePtr val);
    void flush_db(DbHandle dbhandle);
```

```
// 模拟专有接口
RetCode create_sweep_attr(DbHandle db, const SweepAttr *sweepAttr);
RetCode begin_sweep(DbHandle db, double *val);
RetCode end_sweep(DbHandle db);
void set_synchronous_mode(DbHandle db, bool syncMode);

// 转换接口(可选)
RetCode convert_to_vcd(const char* dbName, const char* vcd);
RetCode generate_from_vcd(const char* vcd, const char* dbName);

#ifdef __cplusplus
}
#endif
#endif // WAVEDB_API_H
```