Annex B
Include files

This annex shows the contents of the `svdpi.h` and `svdpi_src.h` include files.

B.1 Binary-level compatibility include file `svdpi.h`

```
/* canonical representation */

#define sv_0   0
#define sv_1   1
#define sv_z   2  /* representation of 4-st scalar z */
#define sv_x   3  /* representation of 4-st scalar x */

/* common type for 'bit' and 'logic' scalars. */
typedef unsigned char svScalar;

typedef svScalar svBit;    /* scalar */
typedef svScalar svLogic;  /* scalar */

/* Canonical representation of packed arrays */
/* 2-state and 4-state vectors, modelled upon PLI's avalue/bvalue */
#define SV_CANONICAL_SIZE(WIDTH) (((WIDTH)+31)>>5)

typedef unsigned int
        svBitVec32;/* (a chunk of) packed bit array */

typedef struct { unsigned int c; unsigned int d;}
        svLogicVec32; /* (a chunk of) packed logic array */

/* Since the contents of the unused bits is undetermined, the following macros
may be handy */
#define SV_MASK(N) (~(-1<<(N)))

#define SV_GET_UNSIGNED_BITS(VALUE,N)\
   ((N)==32?(VALUE):((VALUE)&SV_MASK(N)))

#define SV_GET_SIGNED_BITS(VALUE,N)\
   ((N)==32?(VALUE):\
    (((VALUE)&(1<<((N)1)))?((VALUE)|~SV_MASK(N)):((VALUE)&SV_MASK(N))))

/* implementation-dependent representation */

/* a handle to a scope (an instance of a module or an interface)*/
typedef void* svScope;

/* a handle to a generic object (actually, unsized array) */
typedef void* svOpenArrayHandle;

/* reference to a standalone packed array */
typedef void* svBitPackedArrRef;
typedef void* svLogicPackedArrRef;

/* total size in bytes of the simulator's representation of a packed array */
/* width in bits */
int svSizeOfBitPackedArr(int width);
int svSizeOfLogicPackedArr(int width);
```



```
/* Translation between the actual representation and the canonical
representation */

/* functions for translation between the representation actually used by
   simulator and the canonical representation */

/* s=source, d=destination, w=width */

/* actual <-- canonical  */
void svPutBitVec32   (svBitPackedArrRef   d, const svBitVec32*   s, int w);
void svPutLogicVec32 (svLogicPackedArrRef d, const svLogicVec32* s, int w);

/* canonical <-- actual  */
void svGetBitVec32   (svBitVec32*   d, const svBitPackedArrRef   s, int w);
void svGetLogicVec32 (svLogicVec32* d, const svLogicPackedArrRef s, int w);

/* Bit selects */

/* Packed arrays are assumed to be indexed n-1:0,
   where 0 is the index of least significant bit */

/* functions for bit select */

/* s=source, i=bit-index */
svBit svGetSelectBit(const svBitPackedArrRef s, int i);
svLogic svGetSelectLogic(const svLogicPackedArrRef s, int i);

/* d=destination, i=bit-index, s=scalar */
void svPutSelectBit(svBitPackedArrRef d, int i, svBit s);
void svPutSelectLogic(svLogicPackedArrRef d, int i, svLogic s);


/*
 * functions for part select
 *
 * a narrow (<=32 bits) part select is copied between
 * the implementation representation and a single chunk of
 * canonical representation
 * Normalized ranges and indexing [n-1:0] are used for both arrays:
 * the array in the implementation representation and the canonical array.
 *
 * s=source, d=destination, i=starting bit index, w=width
 * like for variable part selects; limitations: w <= 32
 */
/* canonical <-- actual */
void svGetPartSelectBit(svBitVec32* d, const svBitPackedArrRef s, int i,
                        int w);
svBitVec32 svGetBits(const svBitPackedArrRef s, int i, int w);
svBitVec32 svGet32Bits(const svBitPackedArrRef s, int i); // 32-bits
unsigned long long svGet64Bits(const svBitPackedArrRef s, int i); // 64-bits
void svGetPartSelectLogic(svLogicVec32* d, const svLogicPackedArrRef s, int i,
                          int w);

/* actual <-- canonical */
void svPutPartSelectBit(svBitPackedArrRef d, const svBitVec32 s, int i,
                        int w);
void svPutPartSelectLogic(svLogicPackedArrRef d, const svLogicVec32 s, int i,
                        int w);
```

```
/* Array querying functions */
/* These functions are modelled upon the SystemVerilog array querying
functions and use the same semantics*/
/* If the dimension is 0, then the query refers to the packed part (which is
one-dimensional) of an array, and dimensions > 0 refer to the unpacked part of
an array.*/

/* h= handle to open array, d=dimension */
int svLeft(const svOpenArrayHandle h, int d);
int svRight(const svOpenArrayHandle h, int d);
int svLow(const svOpenArrayHandle h, int d);
int svHigh(const svOpenArrayHandle h, int d);
int svIncrement(const svOpenArrayHandle h, int d);
int svLength(const svOpenArrayHandle h, int d);
int svDimensions(const svOpenArrayHandle h);

/* a pointer to the actual representation of the whole array of any type */
/* NULL if not in C layout */
void *svGetArrayPtr(const svOpenArrayHandle);

int svSizeOfArray(const svOpenArrayHandle); /* total size in bytes or 0 if not
in C
                                       layout */

/* Return a pointer to an element of the array
   or NULL if index outside the range or null pointer */

void *svGetArrElemPtr(const svOpenArrayHandle, int indx1, ...);

    /* specialized versions for 1-, 2- and 3-dimensional arrays: */
void *svGetArrElemPtr1(const svOpenArrayHandle, int indx1);
void *svGetArrElemPtr2(const svOpenArrayHandle, int indx1, int indx2);
void *svGetArrElemPtr3(const svOpenArrayHandle, int indx1, int indx2, int
indx3);

/* Functions for translation between simulator and canonical representations*/
/* These functions copy the whole packed array in either direction. The user is
responsible for allocating an array in the canonical representation. */
/* s=source, d=destination */
/* actual <-- canonical  */
void svPutBitArrElemVec32 (const svOpenArrayHandle d, const svBitVec32* s,
                           int indx1, ...);
void svPutBitArrElem1Vec32(const svOpenArrayHandle d, const svBitVec32* s, int
indx1);
void svPutBitArrElem2Vec32(const svOpenArrayHandle d, const svBitVec32* s, int
indx1,
                           int indx2);
void svPutBitArrElem3Vec32(const svOpenArrayHandle d, const svBitVec32* s,
                           int indx1, int indx2, int indx3);

void svPutLogicArrElemVec32 (const svOpenArrayHandle d, const svLogicVec32* s,
                             int indx1, ...);
void svPutLogicArrElem1Vec32( const svOpenArrayHandle d, const svLogicVec32*
s,
                              int indx1);
void svPutLogicArrElem2Vec32(const svOpenArrayHandle d, const svLogicVec32* s,
                             int indx1, int indx2);
void svPutLogicArrElem3Vec32(const svOpenArrayHandle d, const svLogicVec32* s,
                             int indx1, int indx2, int indx3);
```



```
/* canonical <-- actual  */
void svGetBitArrElemVec32 (svBitVec32* d, const svOpenArrayHandle s, int
indx1, ...);
void svGetBitArrElem1Vec32(svBitVec32* d, const svOpenArrayHandle s, int
indx1);
void svGetBitArrElem2Vec32(svBitVec32* d, const svOpenArrayHandle s, int
indx1,
                           int indx2);
void svGetBitArrElem3Vec32(svBitVec32* d, const svOpenArrayHandle s,
                           int indx1, int indx2, int indx3);

void svGetLogicArrElemVec32 (svLogicVec32* d, const svOpenArrayHandle s, int
indx1,
                             ...);
void svGetLogicArrElem1Vec32(svLogicVec32* d, const svOpenArrayHandle s, int
indx1);
void svGetLogicArrElem2Vec32(svLogicVec32* d, const svOpenArrayHandle s, int
indx1,
                             int indx2);
void svGetLogicArrElem3Vec32(svLogicVec32* d, const svOpenArrayHandle s,
                             int indx1, int indx2, int indx3);

svBit   svGetBitArrElem (const svOpenArrayHandle s, int indx1, ...);
svBit   svGetBitArrElem1(const svOpenArrayHandle s, int indx1);
svBit   svGetBitArrElem2(const svOpenArrayHandle s, int indx1, int indx2);
svBit   svGetBitArrElem3(const svOpenArrayHandle s, int indx1, int indx2, int
indx3);

svLogic svGetLogicArrElem (const svOpenArrayHandle s, int indx1, ...);
svLogic svGetLogicArrElem1(const svOpenArrayHandle s, int indx1);
svLogic svGetLogicArrElem2(const svOpenArrayHandle s, int indx1, int indx2);
svLogic svGetLogicArrElem3(const svOpenArrayHandle s, int indx1, int indx2,
int indx3);

void svPutLogicArrElem (const svOpenArrayHandle d, svLogic value, int indx1,
...);
void svPutLogicArrElem1(const svOpenArrayHandle d, svLogic value, int indx1);
void svPutLogicArrElem2(const svOpenArrayHandle d, svLogic value, int indx1,
                        int indx2);
void svPutLogicArrElem3(const svOpenArrayHandle d, svLogic value, int indx1,
int indx2,
                        int indx3);

void svPutBitArrElem (const svOpenArrayHandle d, svBit value, int indx1, ...);
void svPutBitArrElem1(const svOpenArrayHandle d, svBit value, int indx1);
void svPutBitArrElem2(const svOpenArrayHandle d, svBit value, int indx1, int
indx2);
void svPutBitArrElem3(const svOpenArrayHandle d, svBit value, int indx1, int
indx2,
                      int indx3);

/* Functions for working with DPI context functions */

/* Retrieve the active instance scope currently associated with the executing
   imported function.
   Unless a prior call to svSetScope has occured, this is the scope of the
   function's declaration site, not call site.
   Returns NULL if called from C code that is *not* an imported function. */
```

```
svScope svGetScope();

/* Set context for subsequent export function execution.
   This function must be called before calling an export function, unless
   the export function is called while executing an extern function. In that
   case the export function will inherit the scope of the surrounding extern
   function. This is known as the “default scope”.
   The return is the previous active scope (as per svGetScope) */
svScope svSetScope(const svScope scope);

/* Gets the fully qualified name of a scope handle */
const char* svGetNameFromScope(const svScope);

/* Retrieve svScope to instance scope of an arbitrary function declaration.
 * (Will be either module, program, interface, or generate scope)
 * The return value will be NULL for unrecognized scope names.
 */
svScope svGetScopeFromName(const char* scopeName);

/* Store an arbitrary user data pointer for later retrieval by svGetUserData()
 * The userKey is generated by the user. It must be guaranteed by the user to
 * be unique from all other userKey’s for all unique data storage requirements
 * It is recommended that the address of static functions or variables in the
 * user’s C code be used as the userKey.
 * It is illegal to pass in NULL values for either the scope or userData
 * arguments. It is also an error to call svPutUserData() with an invalid
 * svScope. This function returns -1 for all error cases, 0 upon success. It is
 * suggested that userData values of 0 (NULL) not be used as otherwise it will
 * be impossible to discern error status returns when calling svGetUserData()
 */
int svPutUserData(const svScope scope, void *userKey, void* userData);

/* Retrieve an arbitrary user data pointer that was previously
 * stored by a call to svPutUserData().  See the comment above
 * svPutUserData() for an explanation of userKey, as well as
 * restrictions on NULL and illegal svScope and userKey values.
 * This function returns NULL for all error cases, 0 upon success.
 * This function also returns NULL in the event that a prior call
 * to svPutUserData() was never made.
 */
void* svGetUserData(const svScope scope, void* userKey);

/* Returns the file and line number in the SV code from which the extern call
 * was made. If this information available, returns TRUE and updates fileName
 * and lineNumber to the appropriate values. Behavior is unpredictable if
 * fileName or lineNumber are not appropriate pointers. If this information is
 * not available return FALSE and contents of fileName and lineNumber not
 * modified. Whether this information is available or not is implementation
 * specific. Note that the string provided (if any) is owned by the SV
 * implementation and is valid only until the next call to any SV function.
 * Applications must not modify this string or free it
 */
int svGetCallerInfo(char **fileName, int *lineNumber);
```

B.2 Source-level compatibility include file `svdpi_src.h`

```
/* macros for declaring variables to represent the SystemVerilog */
/* packed arrays of type bit or logic */
```



```
/* WIDTH= number of bits,NAME = name of a declared field/variable */
#define SV_BIT_PACKED_ARRAY(WIDTH,NAME)/* actual definition will go here */
#define SV_LOGIC_PACKED_ARRAY(WIDTH,NAME)/* actual definition will go here */
```