Section 10.1 and A.1.3 ???

Add BNF for the import and export statements based on the following anywhere a function declaration can occur.

> **import** "DPI" [**pure** | **context**] [<*cname*>=] <named_function_proto>;
>
> **export** "DPI" [<*cname*>=] **function** <*fname*>;

Modify the BNF for named_function_proto to permit open arrays (whatever they are) which can only be used with import "DPI" declarations. Relax prototype syntax to permit unnamed arguments for import "DPI" declarations only which implies that pass by name is not supported.

Section 10.6

Add the following as a new sub-section in Section 10

10.6 Import and Export Functions

The syntax for the import and export of functions is:

> **import** "DPI" [**pure** | **context**] [<*cname*>=] <named_function_proto>;
>
> **export** "DPI" [<*cname*>=] **function** <*fname*>;

In both **import** and **export**, *cname* is the name of the foreign function (import/export), *fname* is the SystemVerilog name for the same function. If *cname* is not explicitly given, it will be the same as the SystemVerilog function *fname*. An error will be generated if and only if the *cname* has characters that are not valid in a C function identifier.

Several SystemVerilog functions may be mapped to the same foreign function by supplying the same *cname* for several *fnames*. Note that all these SystemVerilog functions would have identical argument types (as defined below).

For any given *cname*, all declarations, regardless of scope, must have exactly the same type signature. The type signature includes the return type, the number, order, direction and types of each and every argument. Type includes dimensions and bounds of any arrays/array dimensions. Signature also includes the pure/context qualifiers that may be associated with an import definition.

Only one import or export declaration of a given *fname* is permitted in any given scope. More specifically, for an **import**, the **import** must be the sole declaration of *fname* in the given scope. For an **export**, the function must be declared in the

scope where the **export** occurs and there must be only one **export** of that *fname* in that scope.

For exported functions, the exported function must be declared in the same scope that contains the **export** "DPI" declaration. Only SV functions may be exported (specifically, this excludes exporting a class method)

Note that import “DPI” functions declared this way can be invoked by hierarchical reference the same as any normal SystemVerilog function. Declaring a SystemVerilog function to be exported does not change the semantics or behavior of this function from the SystemVerilog perspective (i.e. no effect in SystemVerilog usage other than making this exported function also accessible to C callers).

Imported functions specified as **pure** shall have no side effects; their results need to depend solely on the values of their input arguments. Calls to such functions can be removed by SystemVerilog compiler optimizations or replaced with the values previously computed for the same values of the input arguments. Specifically, a pure function is assumed not to directly or indirectly (i.e., by calling other functions): — perform any file operations — read or write to any SystemVerilog signal other than those passed as its arguments — access any persistent data, like global or static variables. If a pure function does not obey the above restrictions, SystemVerilog compiler optimizations can lead to unexpected behavior, due to eliminated calls or incorrect results being used.

An unqualified imported function can have side effects but may not read or modify any SystemVerilog signals other than those provided through its arguments. Unqualified imports are not permitted to invoke exported SystemVerilog functions.

Imported functions with the **context** qualifier may invoke exported SystemVerilog functions, may read or write to SystemVerilog signals other than those passed through their arguments, either through the use of other interfaces or as a side-effect of invoking exported SystemVerilog functions. Context functions are always implicitly supplied a scope representing the fully qualified instance name within which the **import** declaration was present. (I.e. an import function always runs in the instance in which the import declaration occurred. This is the same semantics as SystemVerilog functions, which also run in the scope they were defined, rather than in the scope of the caller) Import context functions are permitted to have side effects and to use other SystemVerilog interfaces (including but not limited to VPI). However note that declaring an import context function does not automatically make any other simulator interface automatically available. For VPI access (or any other interface access) to be possible, the appropriate implementation defined mechanism must still be used to enable these interface(s). Note also that DPI calls do not automatically create or provide any handles or any special environment that may be needed by those other interfaces.

It is the user's responsibility to create, manage or otherwise manipulate the required handles/environment(s) needed by the other interfaces. (The svGetScopeName and related functions exist to provide a name based linkage from DPI to other interfaces) Exported functions can only be invoked if the current context refers to an instance from which the named function could be called in SystemVerilog by an unqualified function call (i.e. a call to the function with no hierarchical path qualifier). In general this implies that an exported SystemVerilog function is visible from its declared scope to any scope lower down in the hierarchy. Note that this implies that $root functions are visible to all callers. Attempting to invoke an exported SystemVerilog function from a scope in which it is not directly visible will result in a runtime error; how such errors are handled is implementation dependent. If an imported function needs to invoke an exported function that is not visible from the current scope, it needs to change, via svSetScope, the current scope to a scope that does have visibility to the exported function. This is conceptually equivalent to making a hierarchically qualified function call in SystemVerilog. The current SystemVerilog context will be preserved across a call to an exported function, even if current context has been modified by an application. Note that context is not defined for non-context imports and attempting to use any functionality depending on context from non-context imports can lead to unpredictable behavior.