SystemVerilog 3.1
Testbench Donation

Review and Clarification

Presentation to
SystemVerilog EC

September 4, 2002
Agenda

• Testbench and the Verification Challenges

• Walk-through *Clarification Document*
  ▪ Clarifications on Donation Document
  ▪ Resolution for Compatibility with SystemVerilog
  ▪ Address SV-EC Questions

• Follow on Discussion
Testbench and Verification Challenges

- Designs have become more complex
- Shortened time to market
- Testbench and verification infrastructure: Major challenge
- Testbench donation addressing user needs
Sample SOC and Testbench

Testbench Overview

DUT

10/100M Ethernet
1Gb Ethernet
10Gb Ethernet
USB model
Proprietary model

Ethernet MAC
Ethernet MAC
Ethernet MAC
USB
Proprietary Bus Controller

CPU Core
Control Logic
PCI Controller

Memory Controller
Serial Ports
Parallel Ports
GPIO controller
Infrared controller

External Memory
RS232 model
1284 model
GPIO model
IR device model

Protocol checkers for Interface

Interface boundaries

PCI Model

© 2002 Synopsys, Inc. (4)
SystemVerilog EC: September 4, 2002
Sample System with Multiple SOC’s

Packet Switched Bus

SOC 1
- Controller
- CPU
- Switch-Bar
- Memory

SOC 2
- Controller
- Switch-Bar
- I/O

• At System Level Problem is Exacerbated
• Abstractions are Necessary!
Simulation-based Verification Needs

- Stimulus Generation
  - Directed, Random, ATPG, ...

- Checkers
  - Data
  - Protocols

- Structured Connection to Multiple Independent Interfaces
  - Interconnect
  - Clocking Domain
  - Protocol

- Abstract Modeling
  - High-level data structures
  - Dynamic Memory
    - Memory Management
  - Re-entrant Processes
    - Inter-process Synchronization, Control, and Communication
  - Re-usability

→ Testbench
Document Walk Through

• Basic Data Types
• Sequential Control
• Classes, Objects and Methods
  ▪ Memory Management
  ▪ Linked Lists
• Process Control and Synchronization
• Program Block
• Clocking Domains
  ▪ Connecting the DUT
  ▪ Declaration and Semantics
  ▪ Cycle Abstraction
• Signal Operations
Basic Data Types

Testbench donation

• Lexical Conventions
  ▪ Same as SystemVerilog (SV from here on)

• Statement Blocks (1-3; 3.3)
  ▪ Conflict: { and } for execute blocks conflicts with SV3.0
  ▪ Resolution: Adopt SV conventions:
    • Use begin and end for execute blocks
    • SV syntax for task and functions
    • SV-like syntax for new constructs:
      ▪ class name . . . endclass
      ▪ program name . . . endprogram
Basic Data Types
Testbench donation

• Strings (1-4; 3.4)
  ▪ **Clarification:** String literals same as SV.
    Implicit conversion to string type.
  ▪ **Resolution:** Donation will accept both C notation for ASCII character, as well as SV character notation.

• Numbers (1-5; 3.5)
  ▪ **Clarification:** Un-sized literals not extended to LHS like SV:
    \[\text{bit [2:0]=‘1 } \Rightarrow 3'b111.\]
  ▪ **Resolution:** Donation will be same as SV.

• Virtual Port: (1-6; 3.6)
  ▪ **Clarification:** Virtual port is not supported by donation. Disregard.
Basic Data Types
Testbench donation

• Integer (1-7; 3.7.1)
  ▪ Clarification: Integer is a 32-bit signed data-type on all implementations.
  ▪ Integer default value is ‘X.
  ▪ Resolution: Donation integer will behave exactly as SV integer.

• Bit (1-7; 3.7.2)
  ▪ Conflict: In donation bit is 4-state (like SV reg), in SV bit is 2-state.
  ▪ Resolution: Donation will support both bit and reg as in SV.
    plus all SV types: char, shortint, int, byte, longint, logic.
  ▪ Donation will support multi-dimensional packed arrays.

  ▪ Clarification: Declaration is limited to [MSB:0]
  ▪ Resolution: Donation will be same as SV, allow arbitrary indices.

• String (1-8; 3.7.3)
  ▪ Clarification: String default value is special constant null.
User-Defined Data Types
Testbench donation

• Enum (1-10; 3.8.1)
  ▪ **Conflict:** Syntax for declaration is different than SV.
    Enums represent 2-state integer values only.
  ▪ **Resolution:** Donation will support SV syntax and semantics.
  ▪ **Proposal:** Allow declaration as a shorthand for SV typedef (C++).
    
    ```markdown
    enum name {..};  \rightarrow \text{creates enum type called name.}
    same as typedef enum {..} name;
    ```
  ▪ **Clarification:** Enum default value is first element in enumeration.

• Increment/Decrement Operation (1-21;3.8.1)
  ▪ **Proposal:** SV adopt donation’s operators semantics for enums.
    Alternative is to disallow ++ and -- on enums (C++).
• **Enum in Numerical expressions (1-20; 3.11)**

  **Clarification:** Enums can be assigned numerical values (arbitrary expressions) using the run-time `$cast_assign()` system task.

  ```
  function integer $cast_assign(scalar dest_var, scalar source_exp [, CHECK]);
  Example: $cast_assign( enum_var, 12 * 7 );
  ```

  **Proposal:** *Include `$cast_assign()` in the donation.*
User-Defined Data Types
Testbench donation

• Arrays (1-11; 3.8.2):
  ▪ **Conflict:** No slicing of array element [per Verilog 95 rules].
  ▪ **SV** allows slicing of unpacked arrays by any arbitrary number of dimensions:
    ```
    integer a_array[10:0], b_array[1:0];
    b_array = a_array[3:2];
    ```
  ▪ **Resolution:** Donation will support slicing of packed and unpacked arrays, and left-hand-side assignment, same as SV.

• Array Initialization (1-12; 3.8.3):
  ▪ **Clarification:** No concatenation or replication in array initialization.
  ▪ **Resolution:** Donation will adopt SV syntax and semantics.
User-Defined Data Types

Testbench donation

- **Multi-dimensional Arrays** (1-13; 3.8.4):
  - **Conflict**: Single number denotes the number of elements per dimension (like in C).
  - **Resolution**: Donation will accept SV notation for both packed and unpacked arrays.

- **Proposal**: Accept donation’s syntax as a shorthand notation.

```
int Array[8][32];  =>  int Array[7:0][31:0];
```
User-Defined Data Types

Testbench donation

• Associative Arrays (1-16; 3.9):
  ▪ **Clarification:** This section was omitted from donation.
  ▪ Associative arrays can be declared with a specific index type or with no index type.

  ▪ No index in declaration:  `reg reg_array[];`
    The array can be indexed by any integral data type of any size.

  ▪ Index type in declaration: restricts index expression to that type:
    - **string:**  `reg string_array[string];`
    - **object:**  `reg object_array[myClass];`
    - **integer:**  `reg integer_array[integer];`

  ▪ **Clarification:** Not for arbitrary user-defined types, such as `reg[21:2].`
  ▪ **Resolution:** Donation will support any user-defined type that was defined via typedef.
User-Defined Data Types

Testbench donation

• Dynamic Arrays (; 3.10):
  ▪ **Clarification:** This is missing in the donation.
  ▪ Declaration Syntax:
    bit [3:0] nibble[*];
    integer mem[*];
  ▪ Functions used with dynamic arrays:
    new[] // allocates storage
    mem = new[20];

    int $get_array_size(); // gets size of array

  ▪ **Proposal:** Include dynamic arrays in the donation.
Operators
Testbench donation

• Operators (1-22; 3.12)
  ▪ Clarification: Operators in donation are same as SV, with few exceptions.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>{}</code></td>
<td>LHS numeric concatenation</td>
<td>Same as LHS <code>{}</code> in SV</td>
</tr>
<tr>
<td>`{}</td>
<td>String concatenation</td>
<td>Not in SV</td>
</tr>
<tr>
<td><code>{}</code></td>
<td>String replication</td>
<td>Not in SV</td>
</tr>
<tr>
<td><code>==</code></td>
<td>Wild equality</td>
<td>Not in SV</td>
</tr>
<tr>
<td><code>!=</code></td>
<td>Wild inequality</td>
<td>Not in SV</td>
</tr>
<tr>
<td><code>&amp;~</code></td>
<td>Bit-wise NAND</td>
<td>Not in SV</td>
</tr>
<tr>
<td>`</td>
<td>~`</td>
<td>Bit-wise NOR</td>
</tr>
<tr>
<td><code>?:</code></td>
<td>Conditional</td>
<td>Unlike SV</td>
</tr>
</tbody>
</table>

• Conflict: `|~` and `&~` may change existing verilog code
• Resolution: Remove these operators from the donation.
Operators
Testbench donation

- **Clarification:** Following operators are part of SV, not donation.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&lt;&lt;&lt; &gt;&gt;&gt;&gt;</td>
<td>arithmetic shift</td>
<td>sign preserving shift</td>
</tr>
<tr>
<td>&lt;&lt;= &gt;&gt;&gt;=</td>
<td>Compound assignment</td>
<td>arithmetic shift assign</td>
</tr>
<tr>
<td>**</td>
<td>power</td>
<td>exponentiation</td>
</tr>
</tbody>
</table>

- **Resolution:** Donation will support these operators.

- **Operator Precedence (1-23; 3.13)**
  - **Clarification:** Operator precedence is same as SV.

- **Conditional operators (1-27; 3.16)**
  - **Clarification:** The document should read as follows:
    
    ```
    expression1 ? expression2 : expression3
    ```
  - **Conflict:** The conditional operator does not behave the same as SV when it evaluates to x or z.
  - **Resolution:** Donation will support SV semantics.
Operators
Testbench donation

• Side-effect operators: ++, -- (; 3.17)
  ▪ Clarification: Donation defines semantics (example):

    ```
    function integer pre_inc (var integer a); begin    // ++a
        a += 1;
        pre_inc = a;
    end
    endfunction

    function integer post_inc (var integer a); begin  // a++
        post_inc = a;
        a += 1;
    end
    endfunction
    ```

  ▪ Resolution/Proposal: SV adopt the donation semantics.
String manipulation: (1-28; 3.18)

- **Clarification:** A *string* literal is implicitly converted to *string* type in assignment to *string* type variable.

- **Clarification:** Basic String Operators:
  - Str1 = Str2      equality
  - Str1 != Str2     inequality
  - {Str1, Str2,...,Strn}  concatenation
  - {multiplier{Str}} replication
  - Str.method(...) invokes built-in method.
Operators
Testbench donation

• Built-in String Methods: (1-28; 3.18.1)
  ▪ Clarification: This was omitted in donation:
    • str.len()
    • str.putc(integer I, integerc)
    • str.getc(integer I)
    • str.toupper()
    • str.tolower()
    • str.compare()
    • str.icompare()
    • str.substr()
    • str.atoi()
    • str.atohex()
    • str.atooct()
    • str.atobin()
    • str.itoa()

  ▪ Clarification: Does not list support of real numbers.
  ▪ Resolution: atoreal() function is added to the donation.
Operators and expressions
Testbench donation

• Variable assignment: (1-31; 3.20)
  ▪ Conflict: Donation does not support assignment recursion:
    \[ a=b=c; \]
    SV supports a modified form \[ a=(b=c); \]
  ▪ Resolution: Donation Will accept the SV assignment form.

• Expressions and operators: (3:21)
  ▪ Clarification: SV provides for fixed-size, variable position slices:
    \[ \text{[position} +: \text{size}] \text{ and } \text{[position} -: \text{size}]. \]
  ▪ Resolution: Donation will support the SV slices, as above.

• Signed vs. Unsigned (3:22)
  ▪ Conflict: Donation follows Verilog-1995 rule: zero filled. SV requires
    that sign be extended.
  ▪ Resolution: Donation will support the SV sign-extension.
Operators and expressions
Testbench donation

• Preprocessor directives (2-4; 4.2)
  ▪ **Conflict:** Donation uses `#` for preprocessor directives but SV uses `'`
  ▪ **Resolution:** Donation will use the same as SV.

• Subroutines (2-5; 4.3)
  ▪ **Conflict:** Donation allows blocking functions, SV does not.
  ▪ **Resolution:** Will disallow blocking functions, like SV.
  ▪ **Conflict:** Default subroutine lifetime is automatic, SV is static.
  ▪ **Resolution:** Will adopt SV default.

• Function Return values (2-8; 4.4)
  ▪ **Conflict:** Donation’s void is a special syntax value, in SV it is a type.
    In donation void is used to discard function return values.
  ▪ **Resolution:** Adopt SV usage of void (a type).
    Use a cast operator to discard function return values.
Operators and expressions
Testbench donation

- Tasks (2-9; 4.5)
  - **Clarification:** Local task is unnecessary, since in SV provides global or module scope.

- return statement (2-10; 4.6)
  - **Conflict:** Donation does not allow return statement from a function, SV does.
  - **Resolution:** Donation will allow SV return form.

- External Declarations (2-13; 4.7)
  - **Conflict:** Donation’s extern keyword conflicts with SV extern usage (in SV interface construct).
  - **Resolution:** Donation will deprecate this use of extern.
Sequential Control
Testbench donation

• case statements (3-3; 5.1)
  ▪ Clarification: Donation does not support unique and priority qualifiers.
  ▪ Resolution: Donation will support unique and priority qualifiers.

• for loops statements (3-6; 5.2)
  ▪ Clarification: SV allows variable declaration following for keyword.
  ▪ Resolution: Donation will support loop variable declaration as in SV.

• break and continue (3-8; 5.3)
  ▪ Clarification: Both have the same semantics as SV.
Class, Objects, and Methods
Testbench donation

- Class and Objects (7-1;)
  - **Clarification**: Collection of data and methods that operate on data.

```c
/*
 * Class contains data and methods
 */

class <class_name>
  <data>
  <methods>
endclass
```

- Constructors (7-5; 9.2)
  - **Clarification**: Every class has a built-in \textit{new} method, which calls the parent class constructor and then initializes each member of the current object to its default value. Users can override the new method.
Class, Object Methods

Testbench donation

- **External Classes (7-11; 9.3)**
  - **Clarification:** This use of extern is deprecated (stated previously).

- **Typedef (7-11; 9.4)**
  - **Clarification:** Donation uses *typedef* keyword only for forward-referencing of class declaration. SV uses *typedef* to define user defined types.
  - **Proposal:** *typedef* in SV will allow forward references of classes.

- **Classes, Struct and Unions (; 9.5)**
  - **Clarification:** Classes are different than SV struct and union.
    - Classes are dynamically created, structs are static.
    - Classes are hierarchically type-compatible, structs are bit-compatible.
    - Classes implement handles, structs do not.
    - Classes cannot be arbitrarily cast.
Class, Object Methods
Testbench donation

• Automatic Memory Management(; 9.6)
  • Clarification: Objects are dynamically allocated and reclaimed. Automatic Memory Management provides:
    • Automatically memory reclamation.
    • Eliminates dangling references, and premature deallocation.
    • Prevents memory leaks and crashes.

• Inheritance (; 9.7)
  ▪ Clarification: Class inheritance section is missing in donation.
    Adds several keywords:
    • extends, virtual, protected, super
  ▪ Clarification: The donation supports single inheritance.
Class, Object Methods

Testbench donation

- **super (; 9.7.3)**
  - **Clarification:** The super keyword is used from within a derived class to refer to the parent class.

- **Constructor chaining (; 9.7.5)**
  - **Clarification:** To call a parent’s class constructor from a derived class constructor requires that to be the first executable statement.

- **Assigning objects to other objects: casting (; 9.7.4)**
  - **Clarification:** A derived class can be directly assigned to any of its super-classes.
  - **Clarification:** A super-class can be assigned to its derived classes only through the $cast_assign() system function:
    ```
    $cast_assign( derived_class, super_class );
    ```
  - This is a run-time cast (like C++ dynamic_cast).
Class, Object Methods
Testbench donation

• Encapsulation and data hiding (; 9.7.6)
  ▪ **Clarification:** Classes can have members that are:
    • public (default)
    • local
    • protected

**Clarification:** Class semantics are similar to C++
Class, Object Methods
Testbench donation

• Methods can also be:
  ▪ public
  ▪ local
  ▪ protected
  ▪ virtual

• Virtual Methods, polymorphism, and abstract classes (; 9.7.7)
  ▪ **Clarification**: Virtual class methods enable polymorphism.
    ▪ Abstract class defines the inter-connect or protocol, and the derived classes provide the various implementations.

• Dynamic Method Lookup (; 9.7.7)
  ▪ **Clarification**: Virtual methods are resolved at run-time.
Linked Lists
Testbench donation

• Linked Lists (8-1; 10)
  ▪ **Clarification:** The linked list package is like the C++ STL List container.

• List macros (8-2; 10.1)
  ▪ **Clarification:** Only one macro is needed: MakeVeraList(type).
Concurrency Control
Testbench donation

- fork and join (4-2; 6.1)
  - **Clarification:** “fork … join none” has the same functionality as process in SV.
  - **Proposal:** SV should adopt “fork … join [any | all | none]” construct. Deprecate process.

![Diagram of fork and join constructs]
Concurrency Control
Testbench donation

• wait_var (4-8; 6.2)
  ▪ **Clarification:** The data-type supported by wait_var is:
    • integer, reg, reg[], enum, or string.

• terminate (4-9; 6.3)
  ▪ **Clarification:** This constructs considers dynamic parent-child relationship of the processes, unlike SV disable.
    
    ```
    for( int j = 1; j < 4; j++ ) begin
      fork
        task_a();
      join none
    end
    ```

• Maximum Threads (4-11; 6.5)
  ▪ **Clarification:** This is a deprecated feature. Please disregard.
Concurrency Control
Testbench donation

• events (1-9; 6.6)
  ▪ **Clarification**: The event definition is omitted.
    event variables are unique data type operated by system tasks
    ▪ $sync similar to @(event_var)
    ▪ $trigger similar to -> event_var
  • By default events are OFF.
  • Events have persistency
  • Tasks sync() and trigger() synchronize statement executions.

  ▪ **Conflict**: Donation event data-type is a superset of SV’s named events.
  ▪ **Proposal**: Extend SV event with donation’s event functionality.
    • Traditional use (@ and ->) is completely backward compatible.
Concurrency Control
Testbench donation

• Event Operations and mechanisms (6.6.1—6.6.3)

  ▪ $sync takes several forms
    • ALL
    • ANY
    • ORDER
    • CHECK

  ▪ $trigger takes several forms:
    • ONE_SHOT (like ->)
    • ONE_BLAST
    • HAND_SHAKE
    • ON
    • OFF
Concurrency Control
Testbench donation

- Events control parallel thread execution
  - Example of ORDER event synchronization.
  - Makes it easy to manage ordered execution of statements in threads.

```
fork
{
    LookupCheck();
    $trigger(HAND_SHAKE,lookupchk_evnt );
}
{ FabricRun(); }
{
    QueueArrange();
    $trigger(HAND_SHAKE, queue_evnt );
}
{
    $sync(ORDER, lookupchk_evnt, queue_evnt);
    interruptMonitor();
}
join none
```
Concurrency Control
Testbench donation

• Semaphore construct: (4-12; 6.7.1)
  ▪ Synchronization primitive for arbitration of shared resources.

  ▪ **Clarification**: $alloc() function creates the semaphore.

  ▪ **Clarification**: $alloc() system function in donation is simplified
    
    function integer $alloc(SEMAPHORE, integer key_count)
Concurrency Control
Testbench donation

- Mailbox construct (4-16; 6.8)
  - Provides interprocess communication: fifo message queue.
  - Accepts any data-type from concurrent threads.

- Allocation of Mailbox (4-16; 6.8.1)
  - Clarification: $alloc function similar to semaphore.
  - Clarification: Function is simplified
    
    ```
    function integer $alloc(MAILBOX);
    ```
Program Block
Testbench donation

- Program construct (2-3; 4.1)
  - **Clarification:** The program block provides:
    - Entry point for test-bench execution.
    - Scope for program data.
    - Separates DUT and test
      - Introduces cycle semantics
Program Block
Testbench donation

- Program construct (2-3; 4.1)
  - Proposal: program connects to design through port list.
  - Port list is the same as a module port list.
    ```
    program  program_name ( list_of_ports );
    program_declarations
    program_code
    endprogram
    ```
  - Ports can be associated with clocking signals: clocking domains
  - cycle-semantics enabled inside the program block.
  - Clarification: Data declarations that include initialization need to be initialized before program execution (same as in SV), but program needs to be called after all other initial blocks.
Program Block
Testbench donation

• Program Scoping Rules (; 4.1.2)
  ▪ Clarification: Following constructs have global scope in donation.
    • Type declarations: class, enum
    • Subroutines
    • Program block
    • Vera interfaces (clocking domain)
    • Data declared outside any block in global scope and program block.

  ▪ Conflict: Scoping rules above are not consistent with SV.
  ▪ Resolution: Allow declarations in program, consistent with SV.

• Multiple programs (; 4.1.3)
  ▪ Clarification: Multiple program instances is compatible with SV.
    Very useful for modeling
Cycle Semantics and Clocking Domain
Testbench donation

- Clock Domain Declaration (; 7.1)
  - **Conflict**: Donation’s `interface` keyword collides with SV `interface`.
  - **Resolution**: Instead of `interface`, the `clocking_domain` construct will be used to specify synchronous interfaces.

- **Clarification**: `clocking_domain` groups set of signals sharing a common clock.

- **Clarification**: Deprecate the `hdl_node` construct (use `=` instead). To specify an XMR, use hierarchical path directly (no quotes). The example in page section 7.1 of Clarification Donation becomes:

```verilog
input [2:0] state PSAMPLE #-1 = top.cpu.state;
```
Cycle Semantics and Clocking Domain
Testbench donation

• Clock Domain Declaration (; 7.1)
  ▪ An example for clocking_domain construct

program test (input phi1, input [15:0] data, output write,
              input phi2, inout [8:1] cmd, input enable);

    clocking_domain cd1
    {
      phi1 CLOCK;
      data PSAMPLE #-1;
      write PHOLD #1;
      input [2:0] state PSAMPLE #-1 = top.cpu.state; //note
    }

    clocking_domain cd2
    {
      phi2 CLOCK;
      cmd NSAMPLE #-2ps NHOLD;
      enable PSAMPLE #1;
    }

    // program begins here
    ...

    // user can access cd1.data , cd2.cmd , etc...
endprogram
Multiple Clocking Domains (; 7.1.1)

- **Clarification**: Ports can be used in more than one clocking domain.
  - Inputs are sampled on the corresponding clock edge.
  - Last outputs drive wins (no resolution).

```verilog
clocking_domain d1
{
    phi1 CLOCK;
    output out1 PHOLD;
}
clocking_domain d2
{
    phi2 CLOCK;
    output out1 PHOLD;
}
```
**Cycle Semantics and Clocking Domain**

Testbench donation

- **Interface signal Declarations (; 7.2)**
  - **Clarification:** Donation does not define I/O skew specification:
    - **Input**
      - NSAMPLE
      - PSAMPLE
      - #-skew_value
    - **Output**
      - NHOLD
      - PHOLD
      - #skew_value

**Resolution:** Skews can be specified in ticks and physical time units.
Cycle Semantics and Clocking Domain

Testbench donation

Cycle Abstraction (; 7.3)

• **Clarification**: Cycle abstractions are enabled by:
  - Clocking domain specification.
  - The *current* or steady state of the system.
    - Requires test-bench execution after all signal propagation
      → After non-blocking assignments.

Execution at this point has several benefits:
  - Avoids non-determinism and race conditions.
  - One consistent model for assertions and test-bench.
  - One consistent view for simulation and formal tools.
  - Fewer module evaluations.

  **Proposal**: Add the execution after NBA to the SV simulation queue.
Cycle Semantics and Clocking Domain
Testbench donation

• HDL path (; 7.4)
  ▪ **Clarification:** hdl path does not have to start at the root. Any hierarchical path that is visible from the program instantiation point will work, as per SV scoping rules.

• CLOCK (6-2;8.1.1)
  ▪ **Clarification:** An alias for the clock signal in a clocking domain.
    • There are no implicit clocks (i.e., SystemClock).
  ▪ **Clarification:** Two or more clocking_domains can share the same clocking signal.
Signal Operations
Testbench donation

• Synchronization (6-2; 8.2)
  ▪ **Clarification**: SystemClock is not required, usage is deprecated.
  ▪ @(CLOCK) is deprecated
    • Explicit interface name and clock @(ram_bus.CLOCK).

• Output Signal Drive (6-5, 6-6; 8.2)
  ▪ **Conflict**: Non-Blocking Drives are not the same as in SV.
  ▪ **Resolution**: Eliminate non-blocking synchronous drives.

• Testbench Input Signal Sample (6-6; 8.3)
  ▪ **Clarification**: Synchronous signal always evaluates to the sampled value.
Signal Operations
Testbench donation

• Asynchronous Operations (6-9; 8.5)
  ▪ **Clarification:** async modifier is not supported, please disregard.

• Synchronous and Asynchronous Domains (6-8; 8.3)
  ▪ **Clarification:** Lack of CLOCK signal in a clocking domain creates an asynchronous domain.
  ▪ All signals in this domain are asynchronous: not sampled or driven at any clock edge.
  ▪ Skew specification, PHOLD, PSAMPLE, etc. are not allowed.

• Sub-Cycle Delay (6-9; 8.6)
  ▪ **Clarification:** Donation does not allow physical time, only tick time.
  ▪ **Resolution:** Remove this limitation, accept both tick time as well as physical time unit.
Testbench donation

• Follow on Discussion