System Verilog v3.1
Testbench Overview

Arturo Salz
Scientist
Synopsys, Inc.

Overview: What's New at a Glance

- New Built-in Types
  - String, Dynamic Array, Associative Array, List
  - Pass by Reference
- Dynamic Processes and Synchronization
  - fork..join [all,any,none,priority]
  - Semaphore and Mailbox primitives
  - Enhanced Events
- Classes
  - Object Oriented
    - Encapsulation, Inheritance, and Polymorphism
  - Handles - *Safe Pointers*
- Dynamic Memory
  - Objects, threads, strings, dynamic and associative arrays
  - Automatically Managed
- Clocking Domain
  - Synchronous Interfaces
  - Cycle-Based Functionality
  - Race-Free Test-Bench

Basic Data Types

- enum
  - Enhanced SystemVerilog-3.0 enumerations
- string
  - Arbitrary length char array. Automatically resized.
- Arrays
  - Dynamic and Associative
- class
  - Object-Oriented.
- mailbox and semaphore
  - Built-in synchronization classes.
- event variables
  - Enhanced events can be assigned & passed as arguments

Dynamic Arrays

- Syntax: ***data_type name*** **[*];**

```
bit [3:0] nibble [*];
integer mem[*];
```

- size() method – Returns current size

```
int j = mem.size;
```

- new [] – allocates and initializes

```
mem = new[100];
mem = new[mem.size * 2] (mem);
```

Array Assignment or Argument

Fixed-size ← { Fixed-Size (same size, static check)
Dynamic (same size, run-time check)

Dynamic ← { Fixed-Size
Dynamic

```
int  A[10:1], B[5:1];
int  D[*] = new[10];
A = D;              // ok
D = A;              // ok
D = B;              // ok    D = new[5] (B);
B = A; B = D;       // error
```

Associative Arrays

- Syntax: ***data_type name [type];***
  data_type name [];

```
integer i_array[];                 // integral index
bit [20:0] array_b[string]; // string index
string ev_array[myClass];    // class index
```

- Built-in methods:
  - **num()**
  - **delete([input index])**
  - **int exists(input index)**
  - **int first/last(output index)**
  - **int next/prev(var index);**

} traversal

Associative Array Assignment or Argument (by Value)

- LHS and RHS must be of compatible type and same index type
- Associative Arrays can NOT be assigned to
  - Fixed-Size Arrays
  - Dynamic Arrays
- and Vice-versa
- Associative Array Passed by Value
  - Local copy of array
  - Same as other arrays

Enumerations

```
typedef enum {red, green, blue} Colors;
Colors c;
c = red;                    // allowed
c = 1 + 1;                  // compiler error
c = Colors'(2 + 3);         // compile-time cast, ok
```

New in SystemVerilog 3.1

```
$cast( c, 1 + 1 )           // run-time cast, ok
if( $cast( c, 27 ) )        // catch error
   $display( "error" );
```

string

- Arbitrary Length Array of **char** (similar to C)
- Automatically Grown (never truncates)
- String Literals Automatically Converted
- Operators:
  - = Assignment
  - == Equality
  - != Inequality
  - < <= > >= Relational
  - {,} Concatenation
  - {{}} Replication
  - *.method* Built-in Methods

Task and Functions

- Default Arguments

```
Declaration:   task  foo( int j = 5, int k = 8 );
Use:     foo();   foo( 5 );   foo( ,8 );   foo( 5, 8 );
```

- Discarding Function Return Value
  - void = some_function(…);
- Pass by Reference

```
Declaration:   task tk( var int[1000:1] ar );
Use:           tk( my_array );     // no & needed
```

Sequential Control

- Enhance **for** loops
  - Multiple initial statements
  - Multiple ending statements

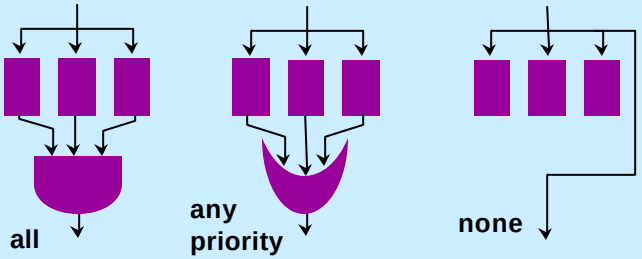
```
for( int count = 0, done = 0, int j = 1;
  j * count < 125;
  j++, count += 3 )
     ...
```

Classes

- Classes contain data and methods
  - **`class myClass data ; methods; endclass`**
  - **`task myClass::method; … endtask`** *//out of body*
- Encapsulation
  - public, local, protected (like C++)
- Inheritance extends classes
  - **`class newClass extends baseClass … endclass`**
- Objects are created and initialized via **new**
  - **`myClass object = new;`**
- Polymorphism supported via virtual methods/classes
  - **`virtual task fun();  virtual class pakt;`**
- Objects Automatically Reclaimed
  - **`object = null;`**

Processes

- Threads can be created anywhere via fork…join
  - all
  - any
  - none
  - priority



- Threads execute until a blocking statement
  - wait for event, mailbox, semaphore, variable, ...
- Thread Control
  - $terminate
  - $wait_child
  - $suspend_thread
  - $exit

Synchronization: semaphore

Built-in class

- new(int n = 0)
  - Create semaphore with *n* keys
- task put(int n = 1)
  - Return *n* keys to the bucket
- task get(int n = 1)
  - Obtain *n* keys from the bucket
- int try_get(int n = 1)
  - Try to obtain a key without blocking

Synchronization : mailbox

Built-in class

- new(int bound = 0)
  - Create a mailbox with a specified queue size
- task put(*message*)
  - Put message in the mailbox queue (FIFO)
- int try_put(*message*)
  - Put message in bounded mailbox queue non-blocking
- task get(var *message*)
  - Retrieve message from the queue
- int try_get(var *message*)
  - Try to retrieve a message without blocking

Parameterized mailbox

- Parameterized built-in class
- **mailbox#(parameter type** = *dynamic*)

```
typedef mailbox#(int) Imbox;
Imbox ib = new();  // int mailbox
int  j;
ib.put( 4 );
ib.get( j );
```

- Compiler Checks Types

Events and Event Variables

Declaration

```
event  a;                  // named event (Verilog)
event  b = null;           // event variable
event  c = a;              // event variable (same as a)
event bit  y;              // persistent event
event bit  q = null;       // persistent event variable
task doit( event e );      // event argument
```

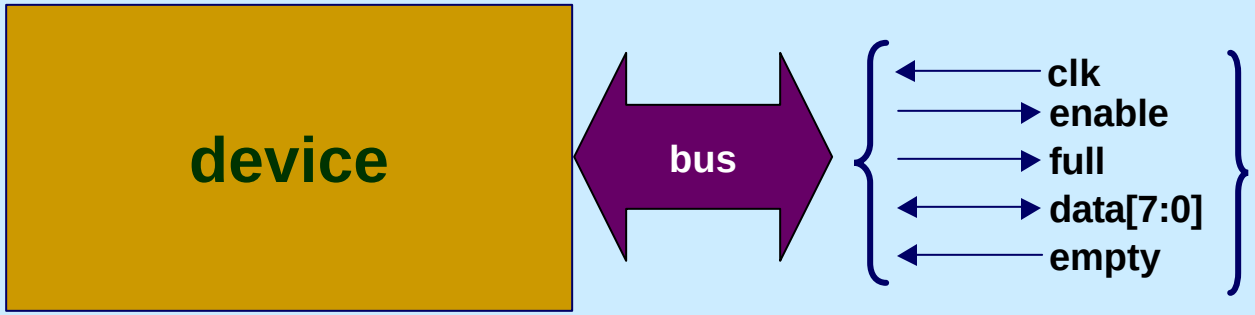
Operations

trigger: -> *event*

synchronize: @ *event*

wait(*event*) *// for persistent*

Synchronous Interfaces: Clocking




```
clocking bus @(posedge clk);
  default input #1ns output #2ns;

  input         enable, full;
  inout         data;
  output        empty;
  output  #6ns  reset = top.u1.reset;
endclocking
```

Clocking Event "clock"

Default I/O skew

Hierarchical signal

Override Output skew

Testbench Uses:

```
bus.enable
bus.data
  ...
```

Default Clocking

- Designate one clocking as default

```
default clocking interface1.bus;
```

- One default per module, interface, or program
- Cycle Delay Syntax: *## integer_expression*

```
## 5;           // wait 5 cycles
## j + 4;       // wait j+4 cycles
```

Program Block

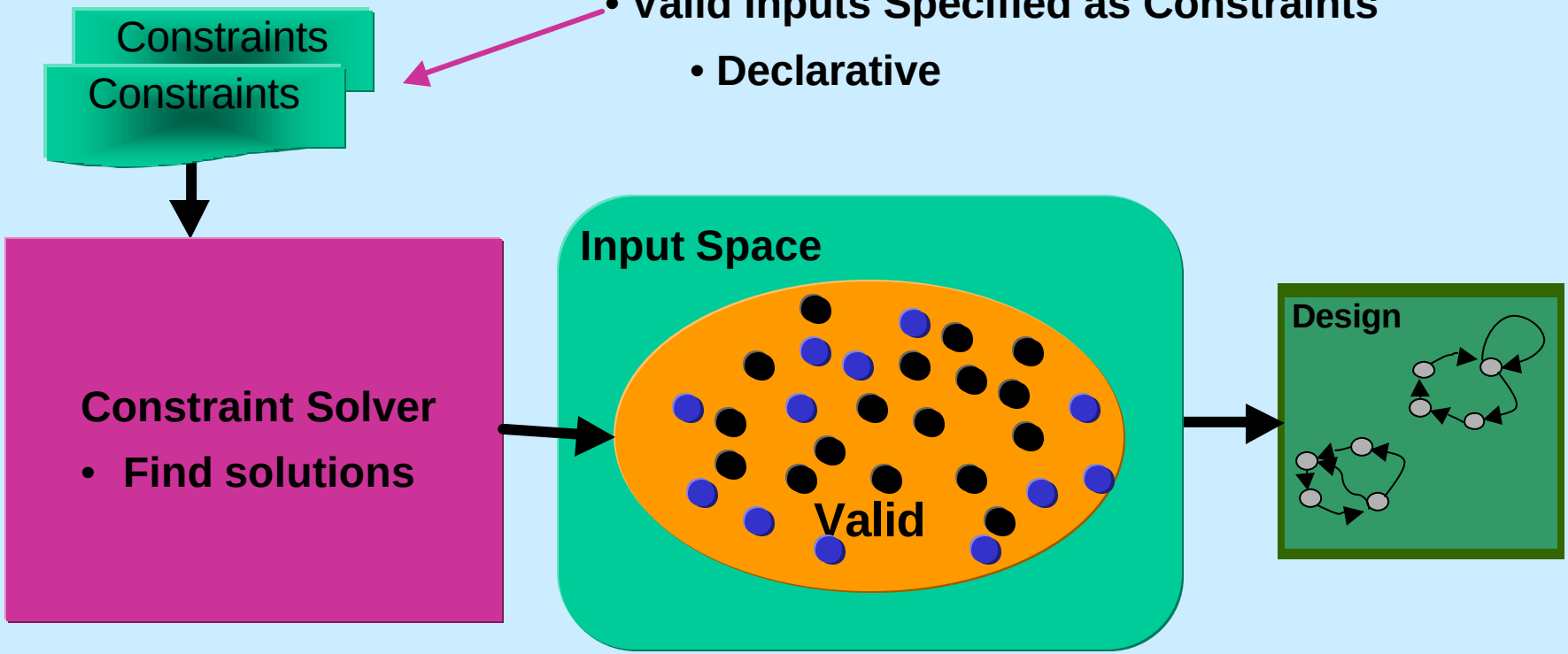
- Purpose: distinguish verification code
- **program** is similar to a **module**
  - Only one implicit initial block
  - Special semantics
    - Execute in verification phase
    - design → clocking → program

```
program name ( port_list );
    declarations (class, type, function, clocking...)
    statements
endprogram
```

Random Constraint Simulation

Test Scenarios

- **Valid Inputs Specified as Constraints**
  - **Declarative**



Random Constraints

- Constraints are built onto the Class system
- Random variables use special modifier:
  - **rand –** random variable
  - **randc –** random cyclic variable
- Object is randomized by calling *randomize( )*
  - Automatically available for classes with random variables.
- User-definable methods
  - **`pre_randomize()`**
  - **`post_randomize()`**

Basic Constraints

```
class Bus;
   rand bit[15:0] addr;
   rand bit[31:0] data;

   constraint word_align { addr[1:0] == 2'b0; }
endclass
```

- Generate 50 data and quad-aligned addresses

```
Bus bus = new;
repeat(50)
 begin
  integer result = bus.randomize();
 end
```

Layered Constraints

```
Typedef enum { low, high, other } AddrType;
class MyBus extends Bus;
    rand AddrType type;
    constraint addr_range
   {
     ( type == low  ) => addr inside { [ 0 :  15] };
     ( type == high ) => addr inside { [28 : 255] };
   }
Endclass
```

- **`type`** variable selects address range
  - Previous constraint (word_align) is still active
  - If *type* is *other*, *addr* is unconstrained
- Use Inheritance to create layered constraint systems

Inline Constraints - *randomize with*

```
task exercise_bus( MyBus bus );
  integer r;
      // restrict to small addresses
  r = bus.randomize() with { type == small; };
      // restrict to address between 10 and 20
  r = bus.randomize() with { 10 <= addr && addr <= 20; };
      // restrict data values to powers of two
  r = bus.randomize() with { data & (data - 1) == 0; };
endtask
```

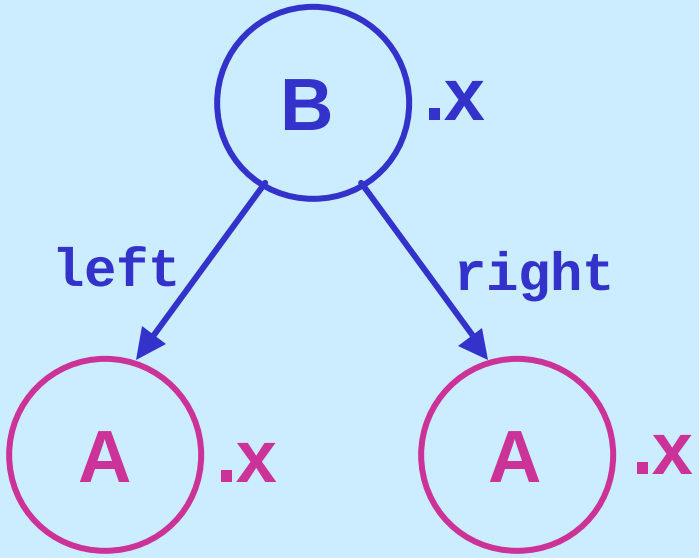
- Constraints are *bi-directional*
  - Including implication (=>)
  - addr depends on type (its constraints) and type depends on addr
- randomize() fails only if there is no solution
  - Conflicting constraints (a > 5; a < 4)

Global Constraints

```
class A ;
   rand   bit[7:0] x;
endclass

class B ;
   rand  A  left;
   rand  A  right;
   rand  bit [7:0]  x;

   constraint btree { left.x <= x ; x < right.x; }
endclass
```



- Aliases are resolved
- Scope rules determine which ‘x’ to use