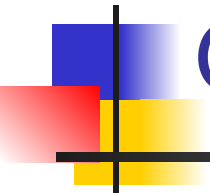


Constrained Random Value Generation

20% 0
80% 1

Probability goes up
can get hard-to-reach-corner cases

Intro

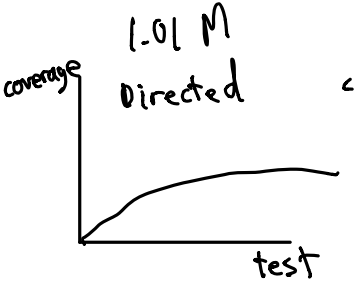
Automatically generate tests for functional verification

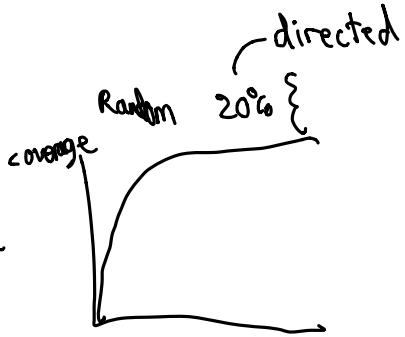
Easily create tests that can find hard-to-reach corner cases.

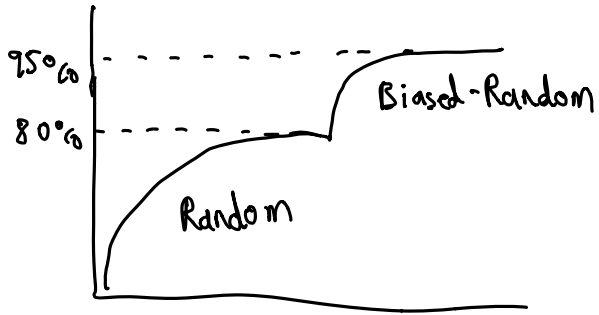
Serial values 50% 0
50% 1

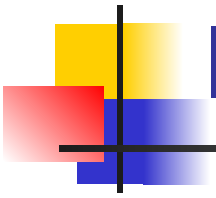
1111 → 111
30-1

DFT stuck at







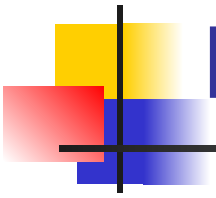


Intro to Classes

class is a type that includes data and subroutines (functions and tasks) that operate on those data.

A class's data are referred to as *class properties*, and its subroutines are called *methods*

We are going to use classes to encapsulate random data



More About Classes

Classes are referenced using an object handle, aka “safe pointer”
Memory allocation is managed internally, garbage collector invoked if no references (similar to Java)
No point arithmetic allowed
Need to use new() to construct/allocate
User-defined methods

Simple Example

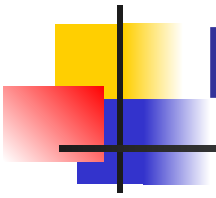
```
class DATA_t;
   rand logic [7:0] addr;
   rand logic [7:0] data;
   rand logic rw;
endclass

DATA_t data; // create a handle
             // data = NULL
data = new; // initialize
```

two types of randoms:
rand
randc → no dups until every random possibility generated

randc[3:0]
No dups until all 16 values generated

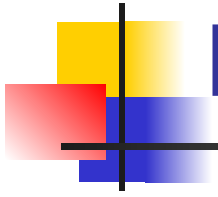
↑ creates the object

data = NULL ← removes the object



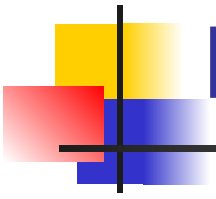
Random Variables

- rand
  - Uniformly distributed over range
- randc
  - Random cyclic that cycle over all possible values



Handle vs Pointer

Operation	C pointer	SV object handle
Arithmetic operations (such as incrementing)	Allowed	Not allowed
For arbitrary data types	Allowed	Not allowed
Dereference when `null`	Error	Error, see text above
Casting	Allowed	Limited
Assignment to an address of a data type	Allowed	Not allowed
Unreferenced objects are garbage collected	No	Yes
Default value	Undefined	`null`
For classes	(C++)	Allowed



Different ways to construct

DATA_t data;
initial data = new;

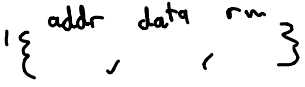
Or

DATA_t data = new; ← implicit initial block

Calling build-in methods

```
module test();
DATA_t data;  // create object handle
initial begin
  data = new();
  $display("%p",data);
  data.randomize();  // built-in method
  $display("%p",data);
  data.randomize();
  $display("%p",data);
end

endmodule
```

prints properties



Constraint Block

Limiting addr/data to some range
constraint low_address { addr <= 8'h55; } address always less than hex 55
constraint low_data { data <= 8'h55; } data always less than hex 55

Set Membership
constraint corner1 addr is random between specified ranges
{addr inside
{[0:3], [70:80], [250:255]};}

Weighted Distribution
x dist {100 := 1, 200 := 2, 300 := 5}

Get value 300 is 5x more likely than 100

constraint equal {
addr == data; }
constraint not equal
{addr != data; }

Implication (easy) - 1

```
rand logic [2:0] addr;
rand logic rw;
constraint one {  rw==1'b0 ->  addr < 4; }
```

if rw is 0, I want address to be generated as less than 4

1st bit is rw

(0,0) (0,1) (0,2) (0,3) ~~(0,4) (0,5) (0,6) (0,7)~~
(1,0) (1,1) (1,2) (1,3) (1,4) (1,5) (1,6) (1,7)

%rw=1 8/12 -> 67%

a |-> b
~a || b
rw=1 or addr<4

Implication (easy) - 2

```
rand logic [3:0] addr;
rand logic rw;
constraint one {addr < 4 -> rw==1'b0 }
```

(0,0) (0,1) (0,2) (0,3) (0,4) (0,5) (0,6) (0,7)
~~(1,0) (1,1) (1,2) (1,3) (1,4) (1,5) (1,6) (1,7)~~

%rw=1 4/12 -> 33%

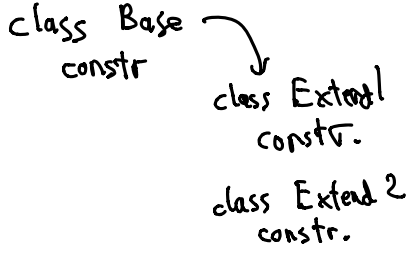
Implication (difficult)

```
 constraint one {  rw==1'b0 ->  addr < 20; }
```

Equivalent to rw==1 || addr<20
addr[7:0],rw -> 512 combinations
Remove (0,21) -> (0,255) from possible values
% of rw=1 -> 93%

```
 constraint one {  addr < 20 -> rw==1'b0 ; }
```

Equivalent to addr>=20 || rw==0
Remove (1,0) to (1,20) from possible values
% of rw=1 -> 47%



If-else (easy) - 1

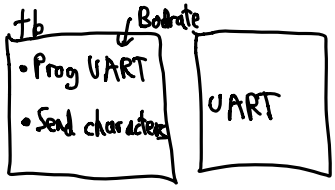
```
rand logic [3:0] addr;
rand logic rw;
constraint one
  {if (addr < 4)  rw==1'b0; }
```

Equivalent to addr < 4 -> rw==1'b0 }

(0,0) (0,1) (0,2) (0,3) (0,4) (0,5) (0,6) (0,7)
(1,0) (1,1) (1,2) (1,3) ~~(1,4) (1,5) (1,6) (1,7)~~

%rw=1 4/12 -> 33%

UART - IP



Randomizing Array

```
class Some_Array;
  rand byte chars[];
  constraint c1 { chars.size inside {[60:90]}; }
  constraint c2 {
     foreach ( chars[ k ] )
           (k < chars.size - 1) ->
           (chars[k] > 33) && (chars[k] < 126); }
endclass
```

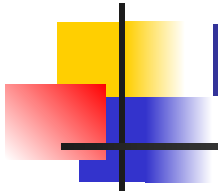
prevents from going out of bounds

$33 \leq chars[k] \leq 126$

constrain between printable ascii characters

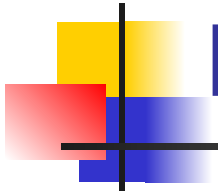
```
byte chars [];
  initial
      char = new[500];
```

Wishbone Bus



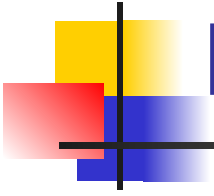
Intro

Open source computer bus
Goal: allow different parts (core) of a chip to communicate with each other in a known, standard way
To enforce compatibility between IP cores. This enhances design reuse
Very simple specification



Features

One Bus Architecture for all applications
Simple, compact architecture
Multi master support
64 bit address space
8 - 64 bit data bus (expandable)
Single read and write cycles
RMW cycles
Event cycles
Supports retry
Supports memory mapped, FIFO and crossbar interface
Throttling of data for slower devices provided
User defined TAGs for identifying data transfer types
Arbitration defined by the end user



Definition

MASTER

- A core that is capable of generating bus cycles
- All Wishbone systems must have at least one MASTER device

SLAVE

- A core that is capable of receiving bus cycles
- All Wishbone systems must have at least one SLAVE device

Interconnect

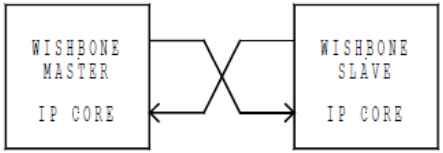


Illustration 1-7: Point to point interconnection.

Point-to-point

An interconnection system that supports a single WISHBONE MASTER and a single WISHBONE SLAVE interface. It is the simplest way to connect two cores.

Crossbar Switch

Support multiple parallel channels which can give high data tr supports n MASTER's

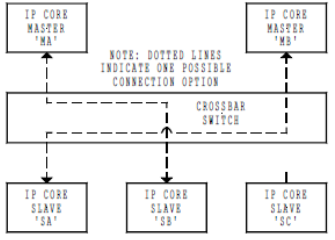


Illustration 1-4: Crossbar (switch) interconnection

Interconnect

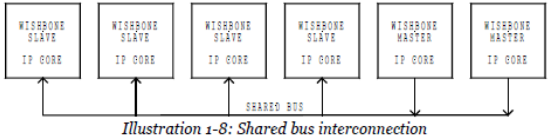


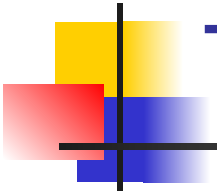
Illustration 1-8: Shared bus interconnection

Shared Bus

The shared bus interconnection is a system where a MASTER initiates addressable bus cycles to a target SLAVE. Only one MASTER at a time can use the interconnection resource

Interface Signals

Signal	Width	Direction	Function
clk_i	1	Input	Clock
rst_i	1	Input	Active high reset
dat_i	32	Input	Input Data bus
dat_o	32	Output	Output Data bus
adr_i	32	Input	Address bus
cyc_i	1	Input	Cycle input, when asserted, indicates that a valid bus cycle is in progress
sel_i	4	Input	Byte select
stb_i	1	Input	Strobe input, when asserted, indicates that the slave is selected
we_i	1	Input	Write enable input, indicates whether the current local bus cycle is a READ or WRITE, asserted during WRITE cycle
ack_o	1	Output	Acknowledge output, when asserted, indicates the termination of a normal bus cycle
err_o	1	Output	Error output (Not Used)
rty_o	1	Output	Retry output, indicates that the interface is not ready to accept or send data and that cycle should be retried (Not Used)



Types of bus cycles

- Single read/write (single data transfer)
- Block read/write (multiple single)
- RMW (read-modify-write)
  - Used in multiprocessor systems
- Handshaking mechanism allows a participating SLAVE to
  - accept a data transfer [ACK_O] *required*
  - reject a data transfer with an error [ERR_O] *optional*
  - ask the MASTER to retry a bus cycle [RTY_O] *optional*

Single READ/WRITE cycle

MASTER generates valid address/data
SLAVE monitors stb_i
SLAVE responds with ack_i

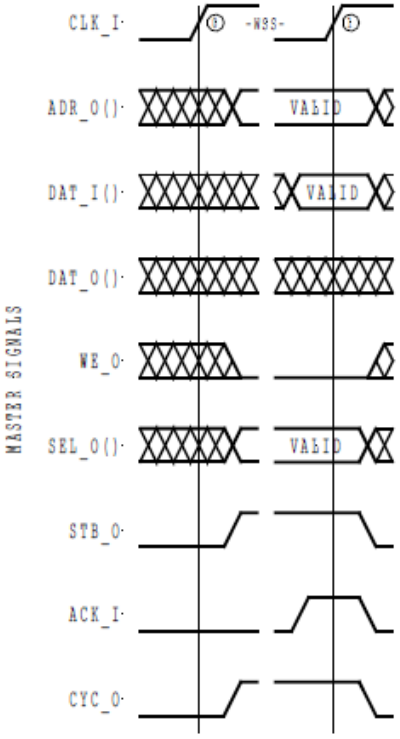
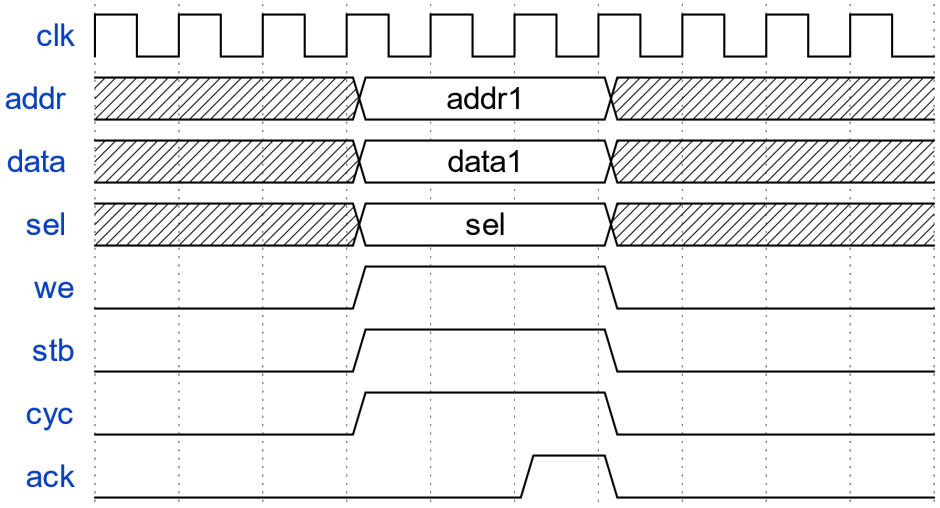


Illustration 8-7: SINGLE READ cycle

WRITE transaction

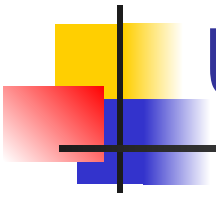


UART top-level

```
module uart_top (
        wb_clk_i,

        // Wishbone signals
        wb_rst_i, wb_adr_i, wb_dat_i, wb_dat_o, wb_we_i, wb_stb_i,
        wb_cyc_i, wb_ack_o, wb_sel_i,
        int_o, // interrupt request

        // UART signals
        // serial input/output
        stx_pad_o, srx_pad_i,

        // modem signals
        rts_pad_o, cts_pad_i, dtr_pad_o, dsr_pad_i, ri_pad_i, dcd_pad_i
);
```

UART interface

```
interface intf_uart_t(input wb_clk_i);

        // Wishbone signals
        logic wb_rst_i;
        logic [4:0] wb_adr_i;
        logic [31:0] wb_dat_i, wb_dat_o;
        logic wb_we_i;
        logic [3:0] wb_stb_i;
        logic wb_cyc_i, wb_ack_o, wb_sel_i;
        logic int_o; // interrupt request

        // UART signals
        // serial input/output
        logic stx_pad_o, srx_pad_i;

        // modem signals
        logic rts_pad_o, cts_pad_i, dtr_pad_o, dsr_pad_i, ri_pad_i, dcd_pad_i;

        clocking wb @(posedge wb_clk_i);
          output wb_rst_i, wb_adr_i, wb_dat_i,wb_we_i,wb_stb_i,wb_cyc_i,wb_sel_i;
          input wb_dat_o,wb_ack_o,int_o;
        endclocking
endinterface
```



UART Tasks

reset()
wb_wr1(addr,sel,data)
Wb_rd1(addr,sel,data)
uart_decoder

skeletal testbench
UART

Assignment

Complete the UART constrained random test
Template located at /home/bgreene/ELEN613/UART

Test out the following
Baud rate
Different #characters sent to uart
Different character sequence

There are 3 “bugs” in the uart, if you discover them, email me for a “fix”

UART TOP Uart_snd()

Put in port definitions
and put constraints

Depending on baud rate & # characters, there are 3 bugs

It will say "Bug Found"

Email Dr. Greene telling which bug (1, 2, or 3)

keycode

simv + BUG1 = 1234

2

3

Read UART Spec on Caming