Stick a *fork* in It: Applications for SystemVerilog Dynamic Processes

Doug Smith and David Long

Doulos

Synopsys Users Group
BOSTON 2010

Processes
Fine-grained process control
Foreign code adapters
Creating stimulus
Summary

APPLICATIONS FOR SYSTEMVERILOG DYNAMIC PROCESSES

DOULOS

What is a “process”?

- A process is a thread of execution
- *Static process* – has a defined end to the execution
  - `assign, initial, always, always_comb`, etc.
  - `fork..join`
- *Dynamic process* – undefined end of execution
  - `fork..join_any` and `fork..join_none`
  - `wait fork, disable fork` and `std::process` for process control
- Useful in system modelling (including testbenches)

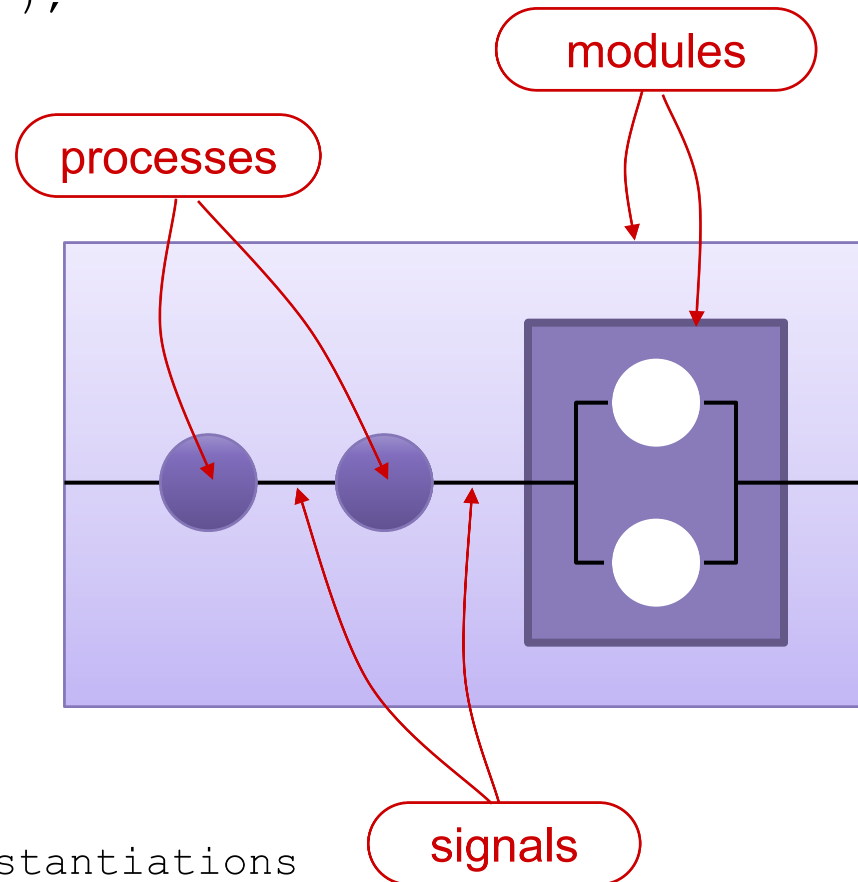
Static processes

```
module Design( input clk, ... );
    // Combinational process
    always @*
    begin
      ...
    end

    // Synchronous process
    always_ff @( posedge clk )
    begin
      ...
    end

    SubBlock b1 ( .* );   // Instantiations

endmodule : Design
```

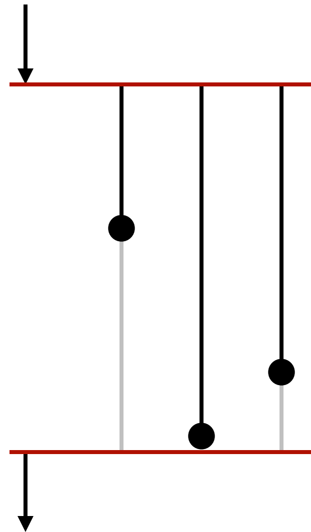


snug
Synopsys Users Group
BOSTON 2010
DOULOS

Static and dynamic processes

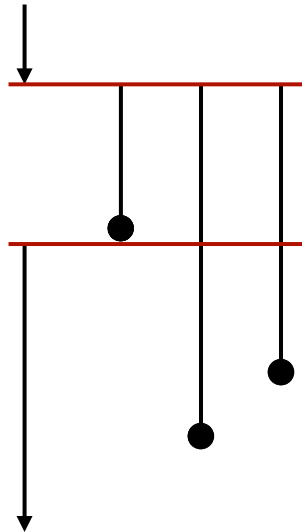
Synopsys Users Group
BOSTON 2010

fork … join



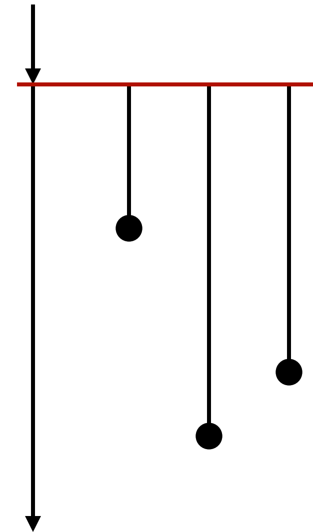
- all branches complete
- invoking code proceeds when all branches finish

fork … join_any



- first branch completes
- invoking code then proceeds

fork … join_none



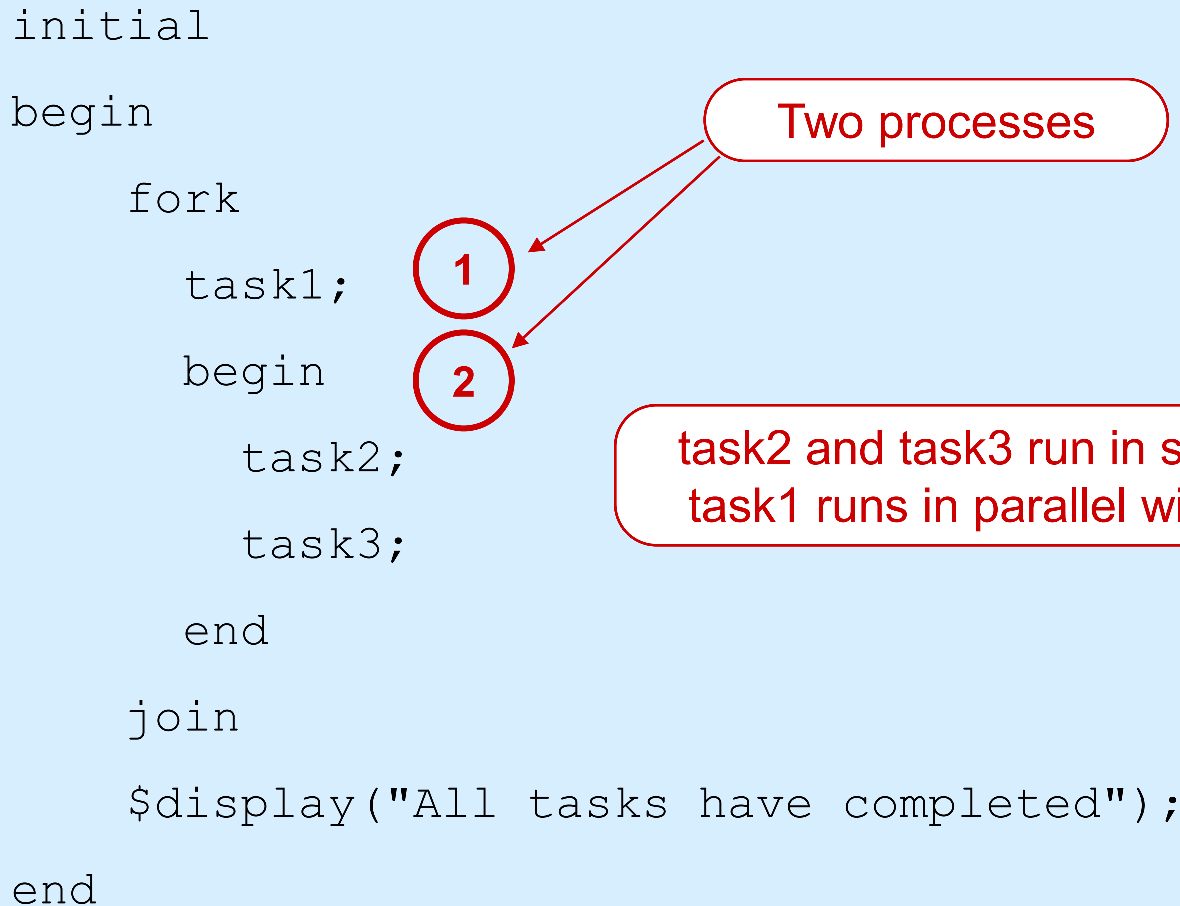
- invoking code proceeds immediately
- all branches run to completion in parallel with invoking code

DOULOS

fork .. join

snug
Synopsys Users Group
BOSTON 2010

- fork .. join blocks until all forked processes have completed

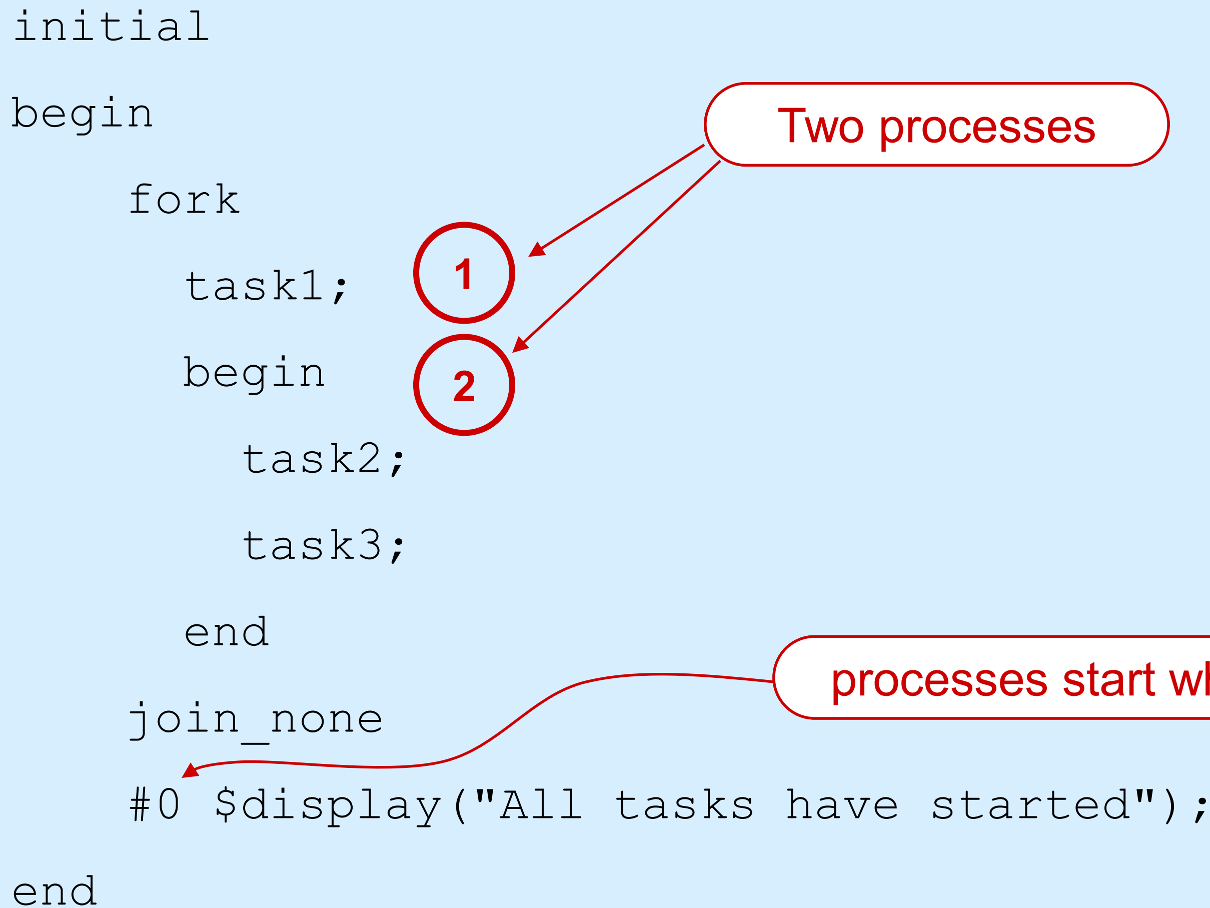


DOULOS

fork .. join_none

snug
Synopsys Users Group
BOSTON 2010

- fork .. join_none spawns “background” processes

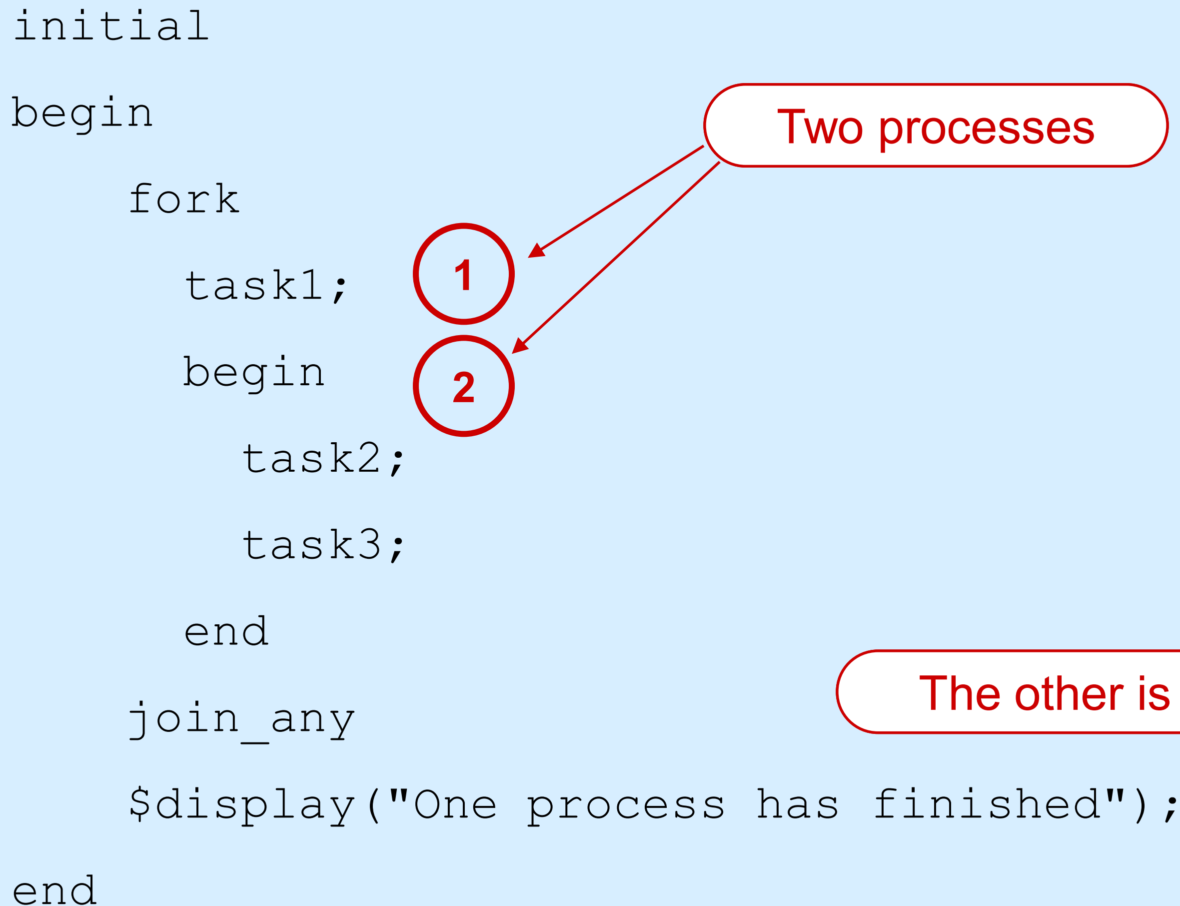


DOULOS

fork .. join_any

Synopsys Users Group
BOSTON 2010

- fork .. join_any also spawns “background” processes



DOULOS

wait fork

- How do I know when processes have completed?

```
initial
  begin
    fork
      ...
    join_none
    ...
    wait fork;
  end
```

This is the parent process

Blocks (waits) until all spawned processes have finished

- Useful to stop tests from exiting before testbench is done

disable kill

Synopsys Users Group
BOSTON 2010

- How can I “kill” sub-processes?

```
initial
  begin
    fork
      ...
    join_any
    disable fork;
  end
```

Blocks (waits) until one spawned process has finished ...

... then kills the rest

DOULOS

Identifying processes

Synopsys Users Group BOSTON 2010

- It is sometimes useful to give processes a unique identifier

```
initial
  begin
    for (int i; i<3; i++)
      fork
        automatic int id = i;
        ...
      join_none
    ...
  end
```

Spawns 3 processes

Local copy of id for each process

- But how do I wait / kill a specific process – *e.g.*, `id` 2?

DOULOS

Fine-grain process control

Synopsys Users Group
BOSTON 2010

- Class `std::process` provides fine-grain process control

```
class process;
enum state {
     FINISHED, RUNNING, WAITING, SUSPENDED, KILLED };
  static function process self();
  function state status();
  function void kill();
  task await();
  function void suspend();
  task resume();
endclass
```

Blocking

kill() or disable fork

Returns handle to current process

- Objects of type `process` are created automatically (cannot call `new`)

DOULOS

Sample applications

- Resolution functions in SystemVerilog
- Creation of independent testbench threads
- Foreign code adapters
- Better generation of stimulus
- Hardware modeling
- Timeouts
- TLI (Transaction Level Interface)
- Architectural modeling

Synopsys Users Group
BOSTON 2010

Processes

Fine-grained process control

Foreign code adapters

Creating stimulus

Summary

APPLICATIONS FOR SYSTEMVERILOG DYNAMIC PROCESSES

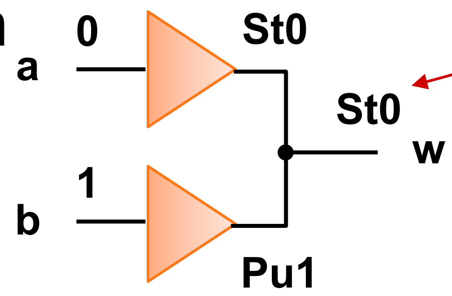
DOULOS

Verilog net resolution

"Higher" strength wins

- Each driver onto a net has a strength

```
assign (strong0, weak1) w = a;
assign (pull0, pull1)   w = b;
```

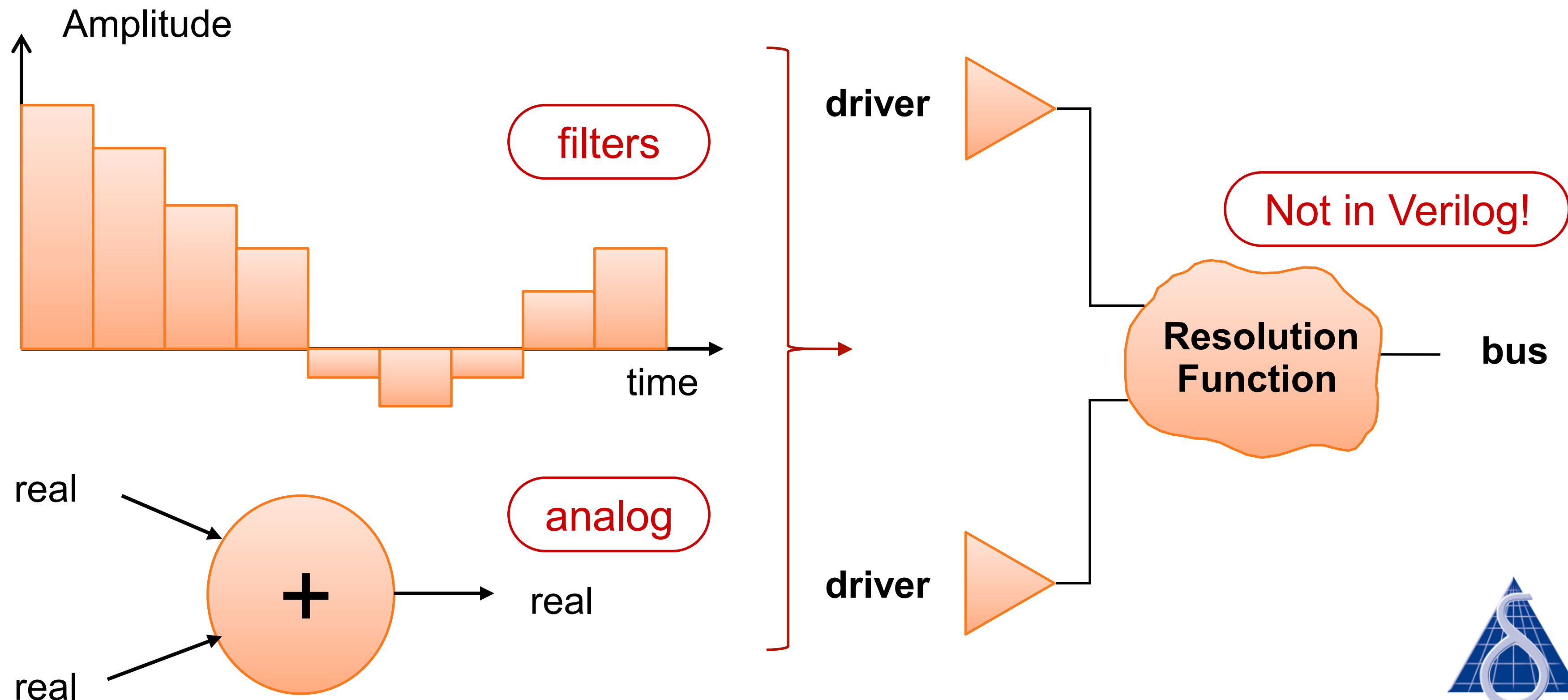


Level	Keyword		Context
7	supply0	supply1	primitive/assign
6	strong0	strong1	primitive/assign
5	pull0	pull1	primitive/assign
4	large		trireg
3	weak0	weak1	primitive/assign
2	medium		trireg
1	small		trireg
0	highz0	highz1	primitive/assign

Default → 6

Custom resolution function?

- `wand` (wired-and) / `wor` (wired-or)
- What about mixed-signal behavior?



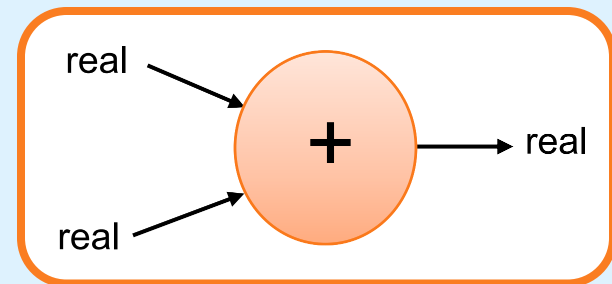
Mixed-signal resolution function

Synopsys Users Group
BOSTON 2010

- Possible in VHDL using resolved type ...

```
package a_pack is
   type real_vec is array (natural range <>) of real;
   function resolve (s: real_vec) return real;
   subtype a_type is resolve real;
end package;
```

```
package body a_pack is
   function resolve (s: real_vec) return real is
     variable sum: real := 0.0;
   begin
       for i in s'range loop
          sum := sum + s(i);
       end loop;
       return sum;
   end function resolve;
end package body;
```



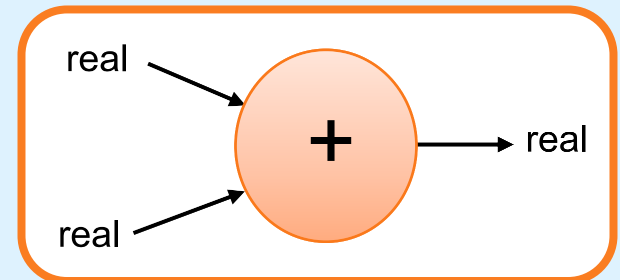
SV Interface resolution function

Synopsys Users Group
BOSTON 2010

```
interface analog_if();
   real driver_vals[process];
   real result;

   function void resolve();
      // Sum all values
      result = 0.0;
      foreach ( driver_vals[proc] )
         result += driver_vals[proc];
   endfunction: resolve

   function void write (real val);
      process this_proc;
      this_proc = process::self;
      driver_vals[this_proc] = val;
      resolve();
   endfunction: write
endinterface: analog_if
```

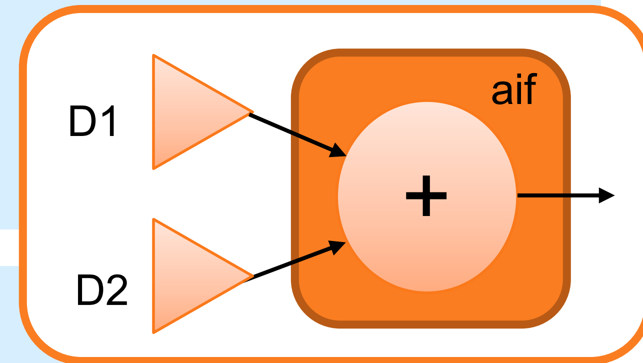


DOULOS

Example

```
module driver #(real low=0.0, high=0.0,
                time delay=1ns)(analog_if aif);
   initial begin
              aif.write(low);
     #delay aif.write(high);
     #delay aif.write(low);
   end
endmodule: driver
```

```
module top;
  analog_if aif1();
  driver #(0.0,2.50,10ns) D1(aif1);
  driver #(0.0,1.25,12ns) D2(aif1);
  ...
endmodule: top
```



Synopsys Users Group
BOSTON 2010

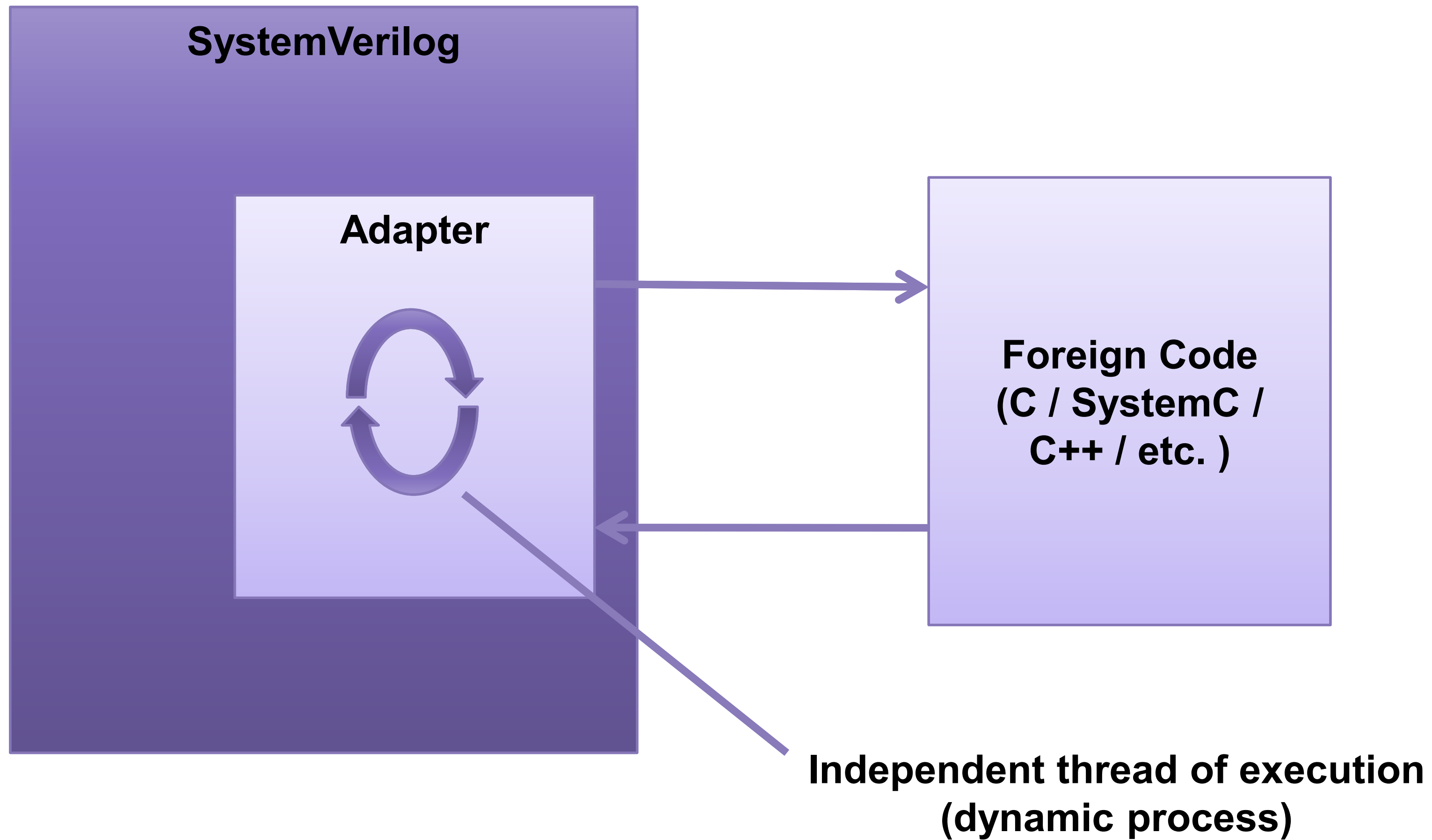
Processes
Fine-grained process control
Foreign code adapters
Creating stimulus
Summary

APPLICATIONS FOR SYSTEMVERILOG DYNAMIC PROCESSES

DOULOS

Foreign Code Adapters

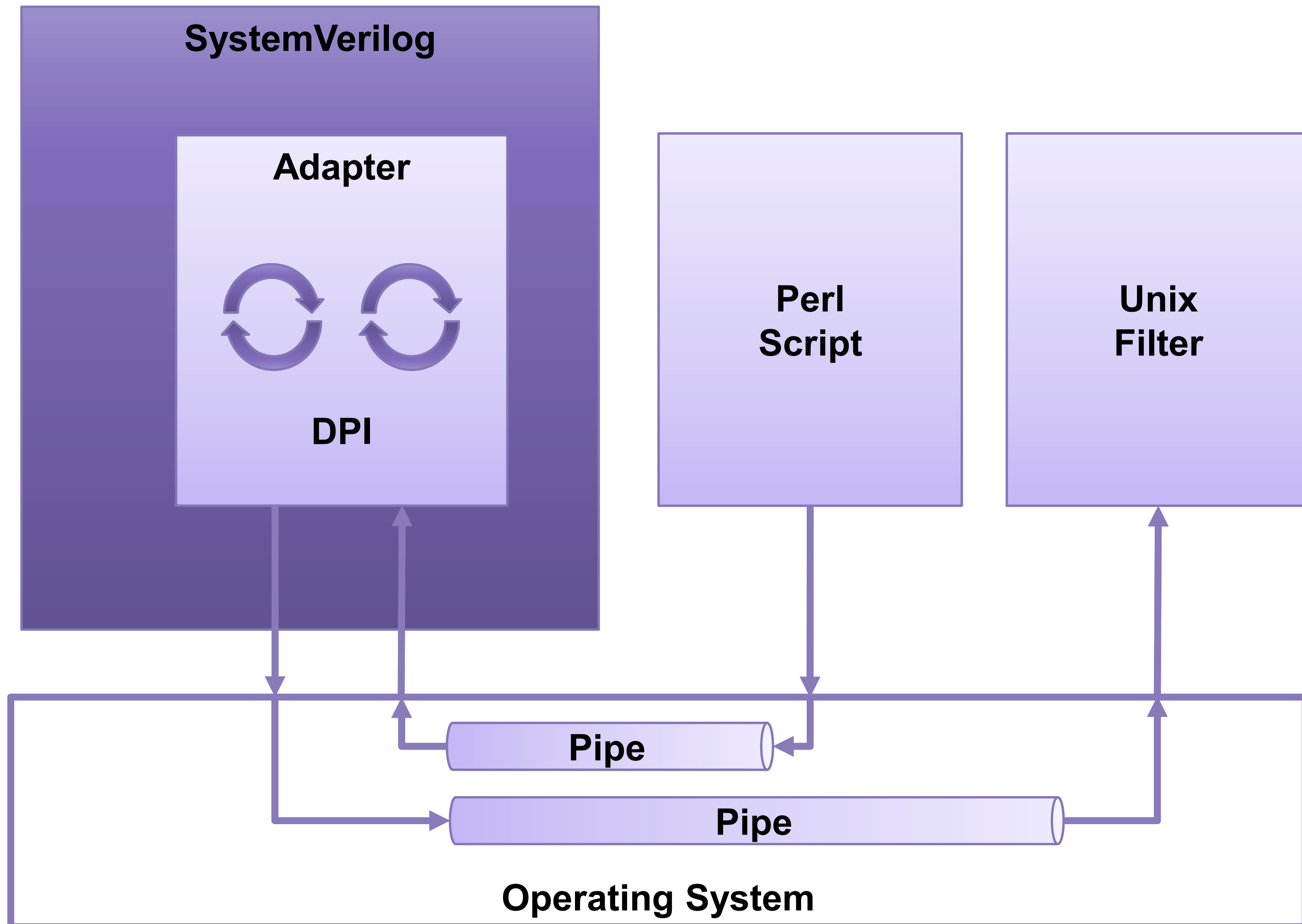
Synopsys Users Group
BOSTON 2010



DOULOS

Example

Synopsys Users Group BOSTON 2010



DOULOS

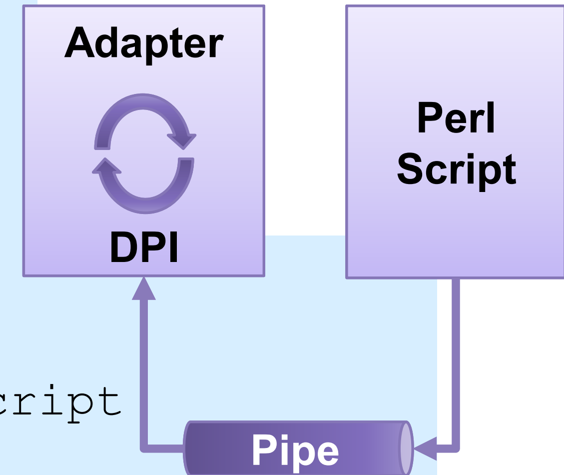
DPI Code (1)

```
#!perl                          # stimgen.pl
@all_ops = ( @operate_ops, @branch_ops, ... );
for ( $i = 0; $i< 100; $i++ ) {
   my $index = rand @all_ops;
   print $all_ops[$index], "\n";
}
```

```
#include <stdio.h>
#include <stdlib.h>
#define PERLCMD "./stimgen.pl"      // Script
#define MAXLINE 256

FILE *fpin;
char line[MAXLINE];

int open_pipe() {
   // Open the pipe to the Perl script
   if ( (fpin = popen( PERLCMD, "r" )) == NULL )
     // Print error if null
     ...
}
```



DPI Code (2)

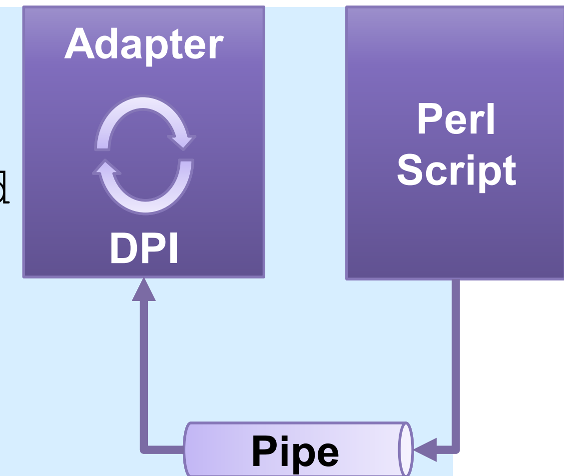
```
char *get_stimulus(){

      // Open pipe if not yet created
      if ( fpin == NULL )
         if ( open_pipe())
            return NULL;

      // Read in the stimulus
      if ( fgets( line, MAXLINE, fpin ) != NULL )
          return line;         // Send to the DPI

      return NULL;
}

// Define Unix filtering functions here ...
```



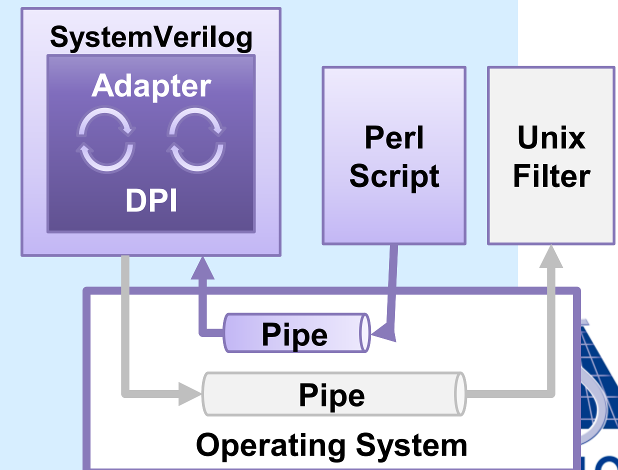
Adaptor code (1)

```
import "DPI-C" function string get_stimulus();
import "DPI-C" function void filter(input string line);
```

```
task run();
    fork
       begin
          string op, in;
          while (( in = get_stimulus()) != "" )
           begin
             assert( $sscanf( in, "%s", op ));
             tr = new ( op );

             // Send transaction
             ...
          end
       end
       ...
```

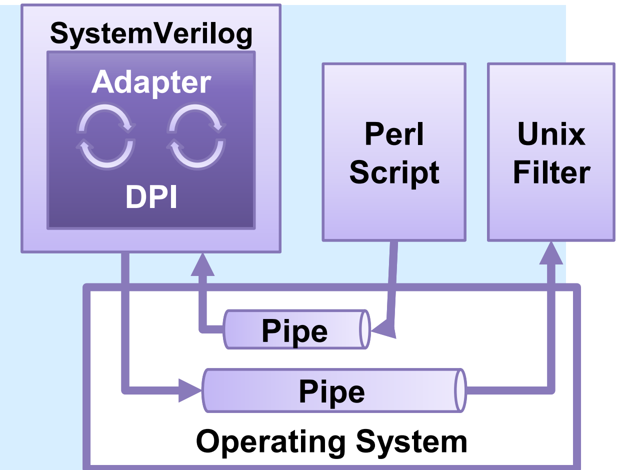
1



Adaptor code (2)

Synopsys Users Group
BOSTON 2010

```
task run();
     ...
     fork
         begin
           // Process perl input
 1         ...
         end
         begin
           forever begin
             @( /* named event */ );
 2           // Call DPI function to send to pipe
             filter( tr.sprint() );
           end
         end
     join_any
     disable fork;
endtask
```



Processes

Fine-grained process control

Foreign code adapters

Creating stimulus

Summary

APPLICATIONS FOR SYSTEMVERILOG DYNAMIC PROCESSES

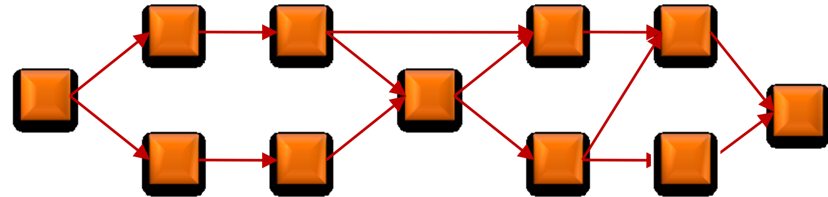
snug
Synopsys Users Group
BOSTON 2010

DOULOS

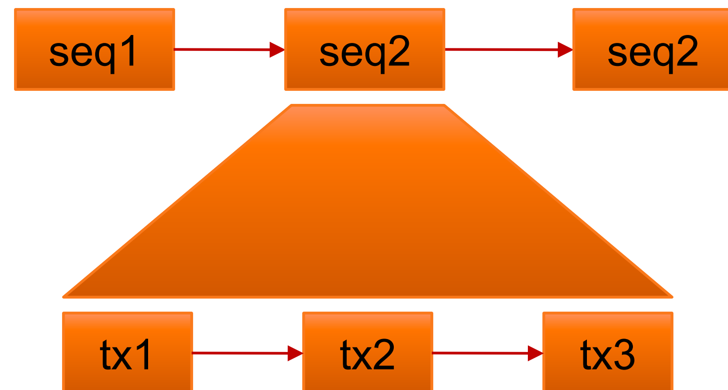
Layered sequential stimulus

Synopsys Users Group
BOSTON 2010

Tests enumerate possible top-level sequences



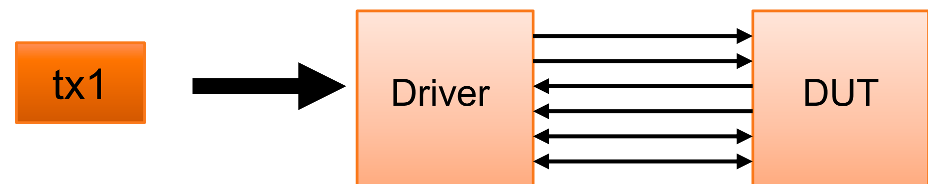
Virtual or layered sequences



Constrained random sequence of transactions

Randomized transactions are not enough

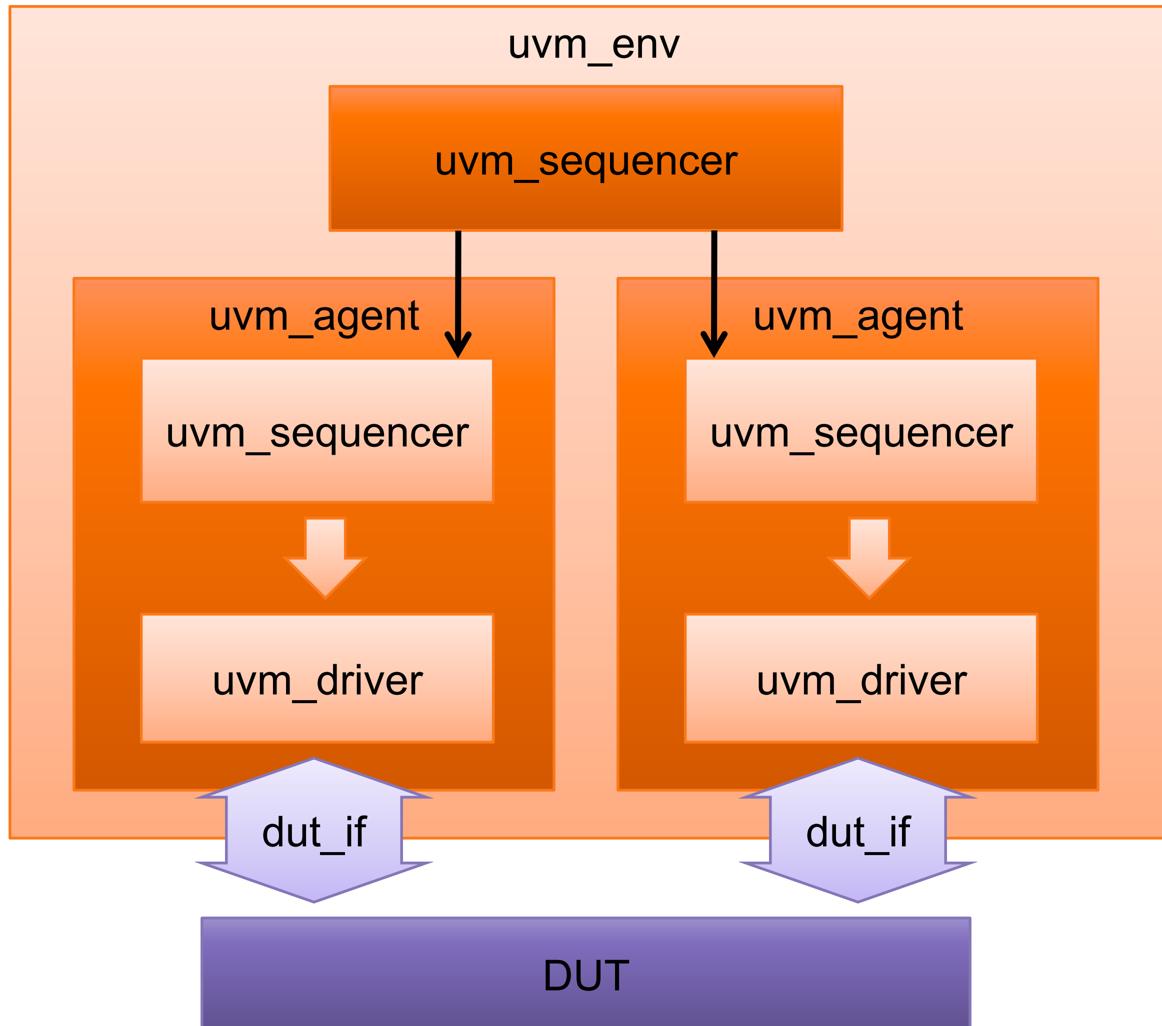
Drive transactions into DUT



DOULOS

Virtual sequences

Synopsys Users Group
BOSTON 2010



DOULOS

Forking sequences

Synopsys Users Group
BOSTON 2010

```
class my_virtual_sequence extends uvm_sequence;

  ...

  virtual task body();
    fork
      p_sequencer.sequencer0.start(iseq_inst0);
      p_sequencer.sequencer1.start(iseq_inst1);
    join_none
    #0;                // Wait allows threads to start

    ...                // Do other work here

    wait fork;

    `uvm_do_on(eseq_inst0, p_sequencer.sequencer0)
    `uvm_do_on(esqe_inst1, p_sequencer.sequencer1)
  endtask: body

endclass: my_virtual_sequence
```

DOULOS

Synopsys Users Group
BOSTON 2010

Processes
Fine-grained process control
Foreign code adapters
Creating stimulus
Summary

APPLICATIONS FOR SYSTEMVERILOG DYNAMIC PROCESSES

DOULOS

Summary

- Dynamic processes and process control enables:
  - Resolution functions in SystemVerilog!
  - Foreign code adapters
  - Better generation of stimulus
- See paper ...
  - Creation of independent testbench threads
  - Hardware modeling
  - Timeouts
  - TLI (Transaction Level Interface)
  - Architectural modeling

Any Questions?

Delivering Know-How

www.doulos.com

Hardware Design

» VHDL » Verilog » SystemVerilog

» Actel » Altera » Xilinx

Embedded Systems and ARM

» C » C++ » UML » RTOS » Linux

» ARM Cortex A/R/M series

ESL & Verification

» SystemC » TLM-2.0 » SystemVerilog

» OVM/VMM/UVM » Perl » Tcl/Tk

All trademarks acknowledged