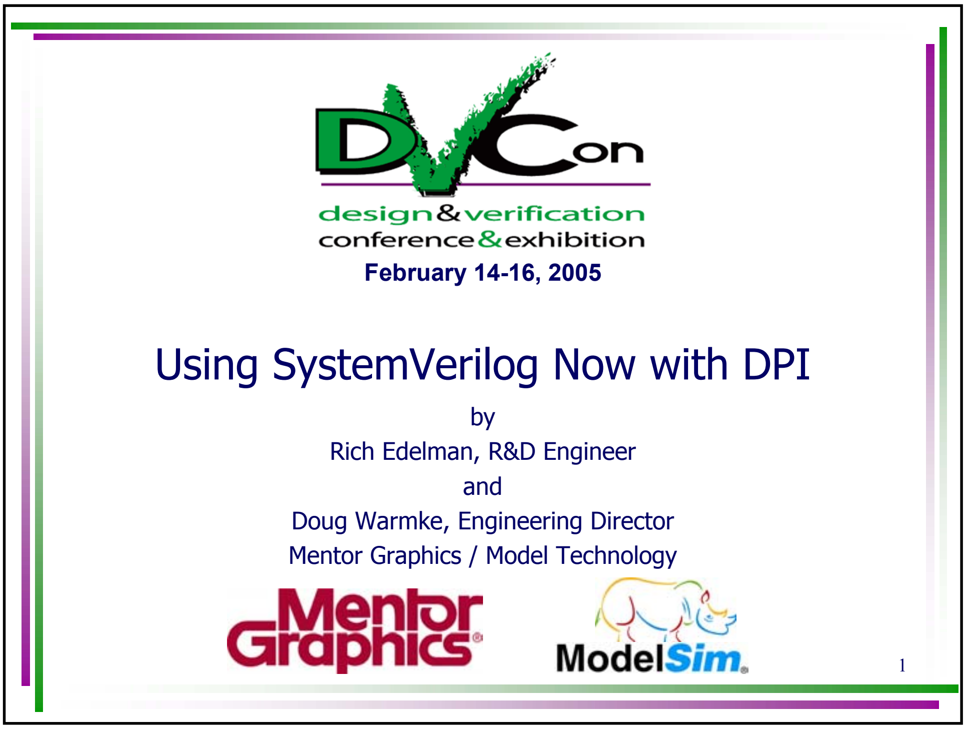
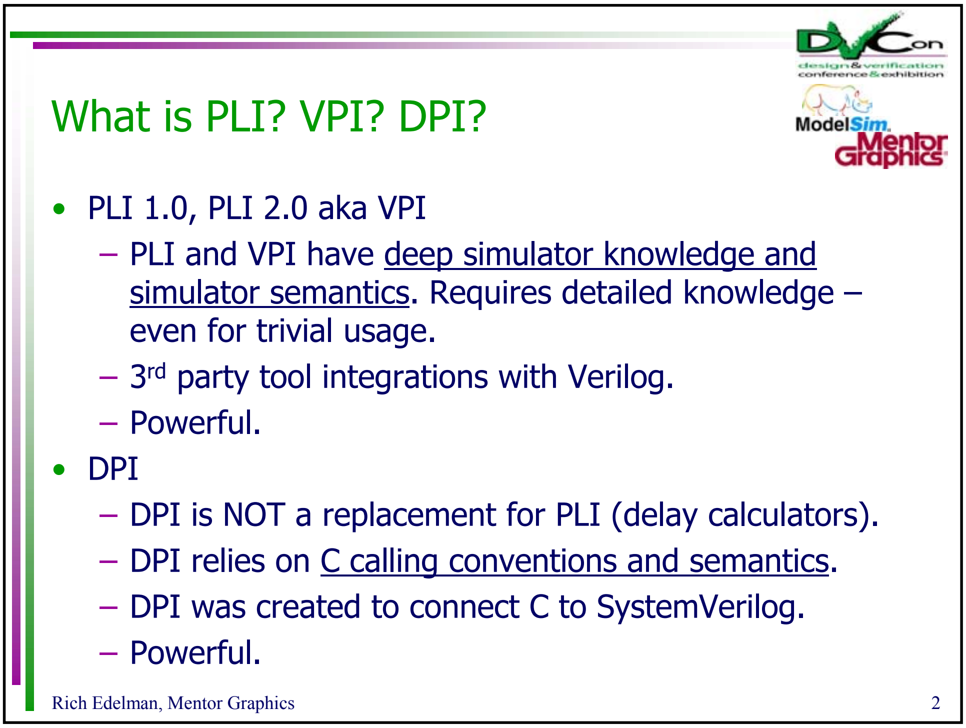


design & verification conference & exhibition

February 14-16, 2005

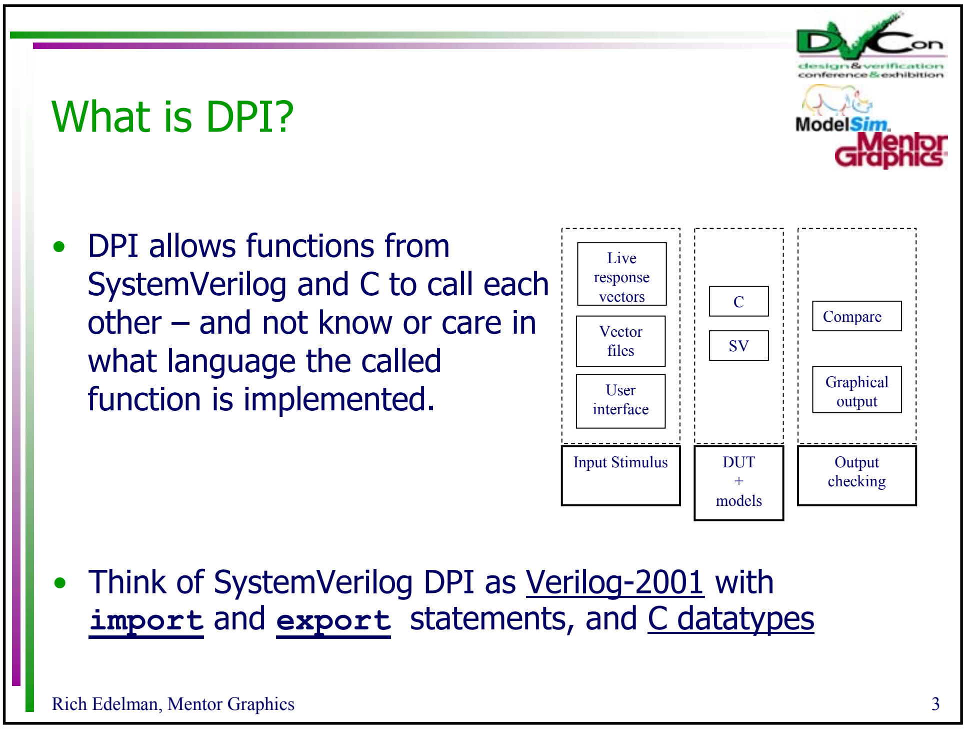
# Using SystemVerilog Now with DPI

by
Rich Edelman, R&D Engineer
and
Doug Warmke, Engineering Director
Mentor Graphics / Model Technology



## What is PLI? VPI? DPI?

- PLI 1.0, PLI 2.0 aka VPI
  - PLI and VPI have deep simulator knowledge and simulator semantics. Requires detailed knowledge – even for trivial usage.
  - 3rd party tool integrations with Verilog.
  - Powerful.
- DPI
  - DPI is NOT a replacement for PLI (delay calculators).
  - DPI relies on C calling conventions and semantics.
  - DPI was created to connect C to SystemVerilog.
  - Powerful.

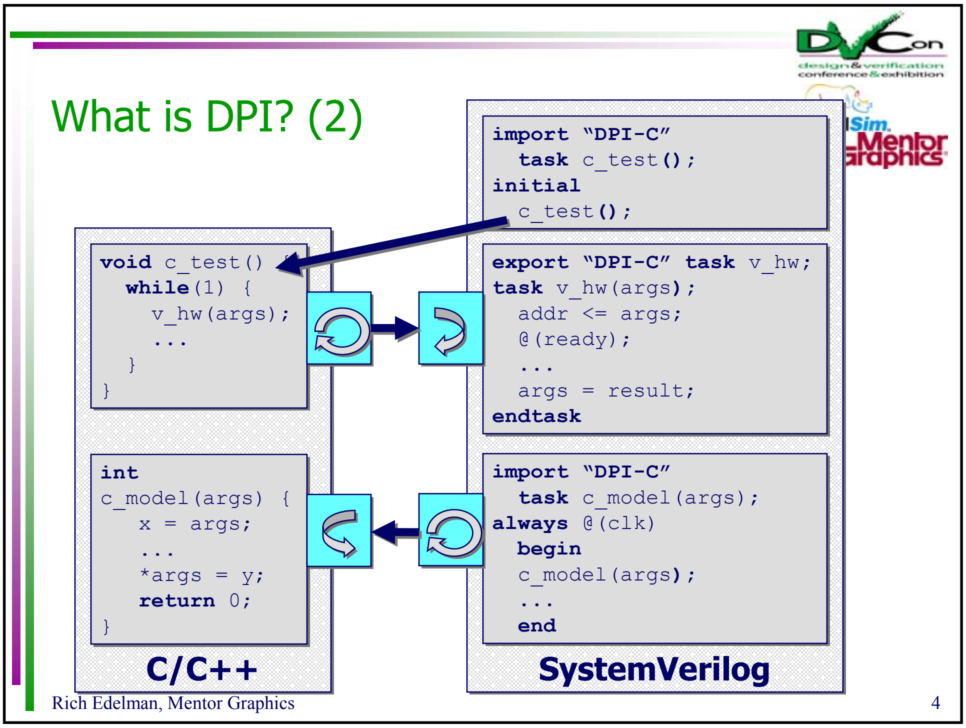


## What is DPI?

- DPI allows functions from SystemVerilog and C to call each other – and not know or care in what language the called function is implemented.

Live response vectors
Vector files
User interface
Input Stimulus
C
SV
DUT + models
Compare
Graphical output
Output checking

- Think of SystemVerilog DPI as Verilog-2001 with **`import`** and **`export`** statements, and C datatypes



## What is DPI? (2)

```
void c_test() {
  while(1) {
    v_hw(args);
    ...
  }
}
```

```
int
c_model(args) {
   x = args;
   ...
   *args = y;
   return 0;
}
```

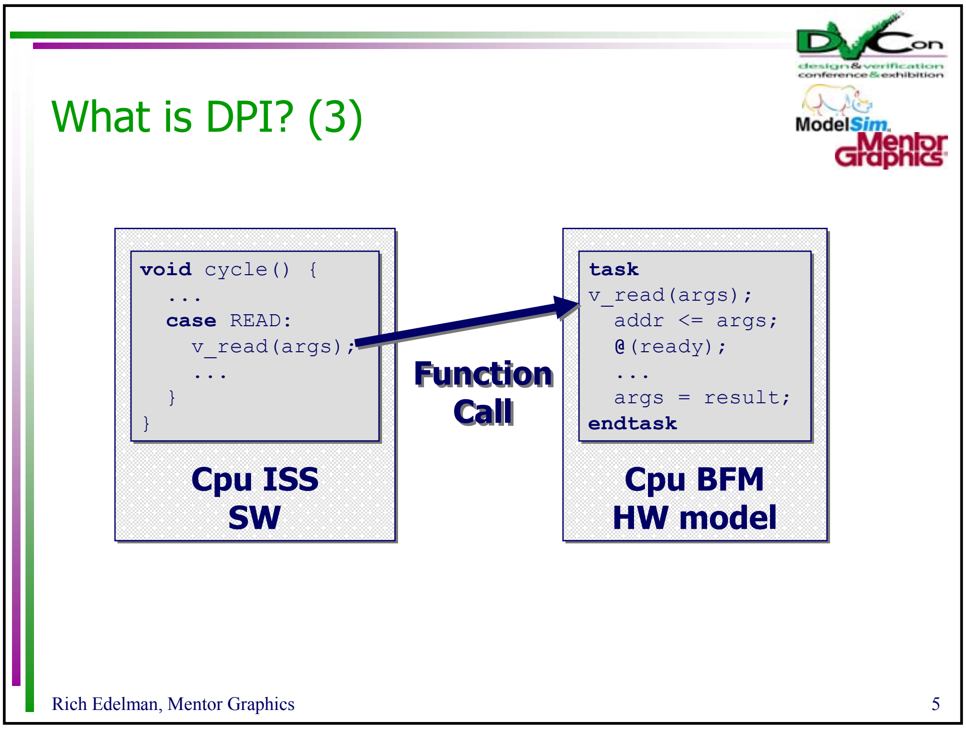
**C/C++**

```
import “DPI-C”
  task c_test();
initial
  c_test();
```

```
export “DPI-C” task v_hw;
task v_hw(args);
  addr <= args;
  @(ready);
  ...
  args = result;
endtask
```

```
import “DPI-C”
  task c_model(args);
always @(clk)
  begin
  c_model(args);
  ...
  end
```

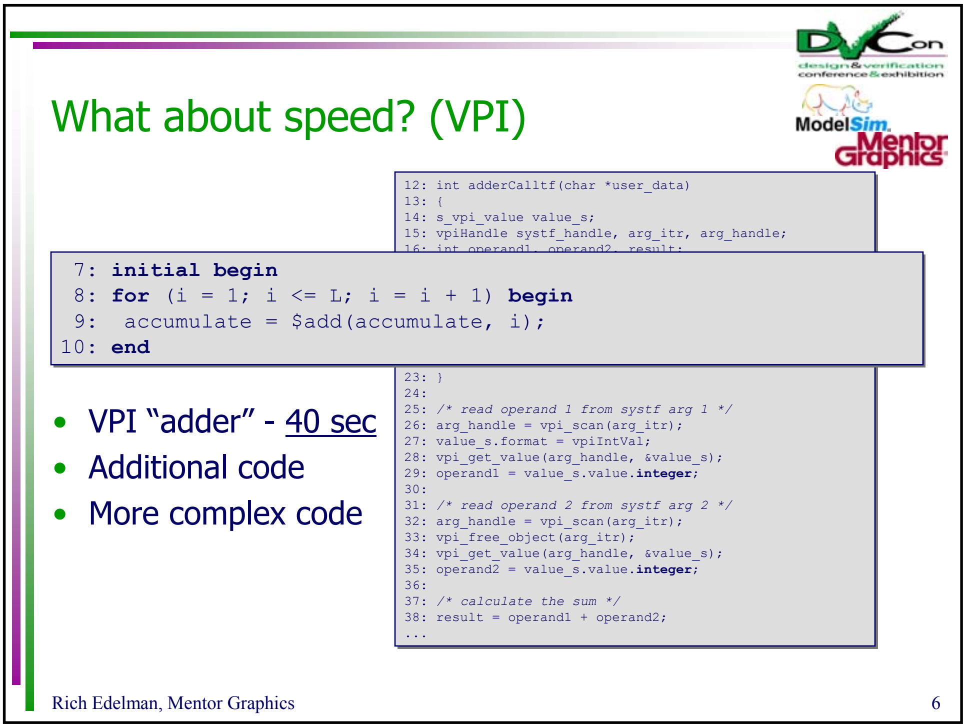
**SystemVerilog**



# What is DPI? (3)

```
void cycle() {
  ...
  case READ:
    v_read(args);
    ...
  }
}
```

```
task
v_read(args);
  addr <= args;
  @(ready);
  ...
  args = result;
endtask
```



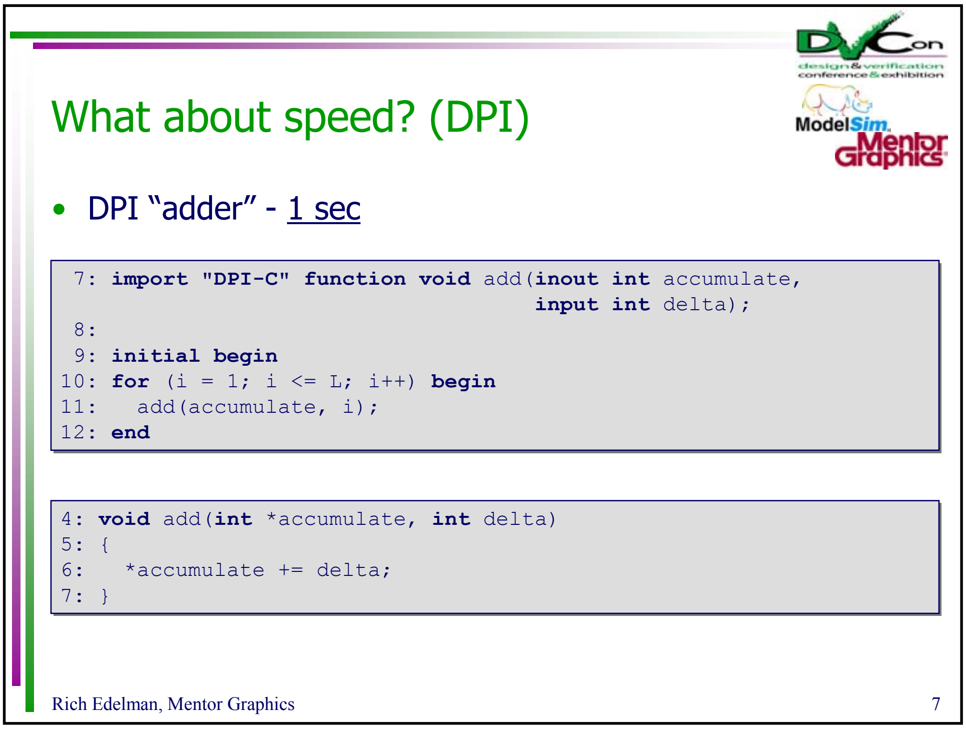
# What about speed? (VPI)

```
initial begin
for (i = 1; i <= L; i = i + 1) begin
 accumulate = $add(accumulate, i);
end
```

```
int adderCalltf(char *user_data)
{
s_vpi_value value_s;
vpiHandle systf_handle, arg_itr, arg_handle;
int operand1, operand2, result;
```

```
}

/* read operand 1 from systf arg 1 */
arg_handle = vpi_scan(arg_itr);
value_s.format = vpiIntVal;
vpi_get_value(arg_handle, &value_s);
operand1 = value_s.value.integer;

/* read operand 2 from systf arg 2 */
arg_handle = vpi_scan(arg_itr);
vpi_free_object(arg_itr);
vpi_get_value(arg_handle, &value_s);
operand2 = value_s.value.integer;

/* calculate the sum */
result = operand1 + operand2;
...
```

- VPI “adder” - 40 sec
- Additional code
- More complex code

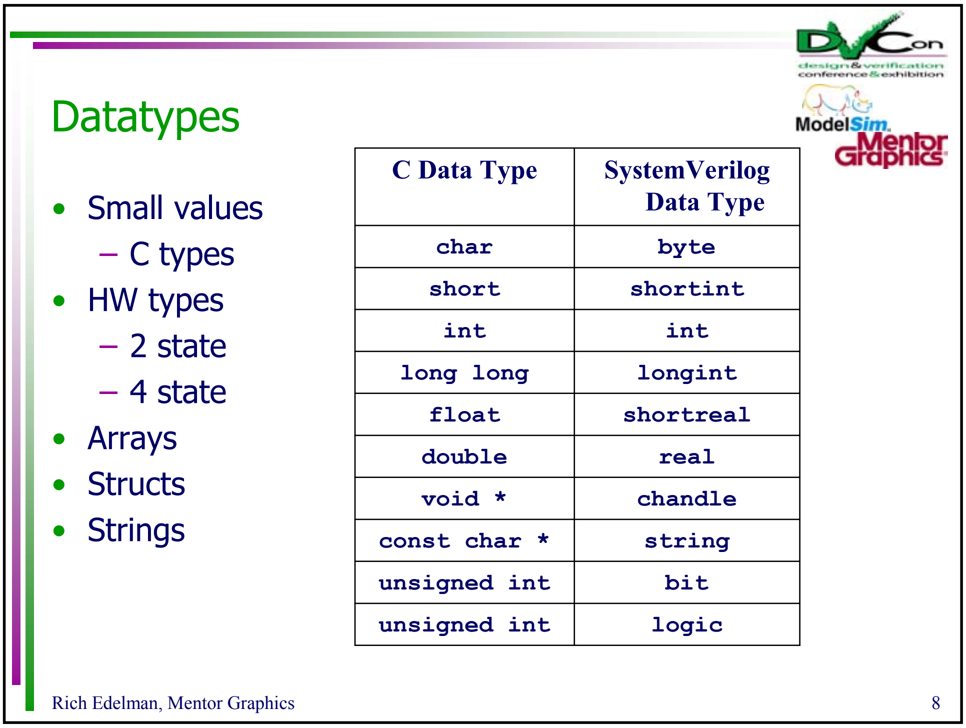


## What about speed? (DPI)

- DPI “adder” - 1 sec

```
import "DPI-C" function void add(inout int accumulate,
                                 input int delta);

initial begin
for (i = 1; i <= L; i++) begin
  add(accumulate, i);
end
```

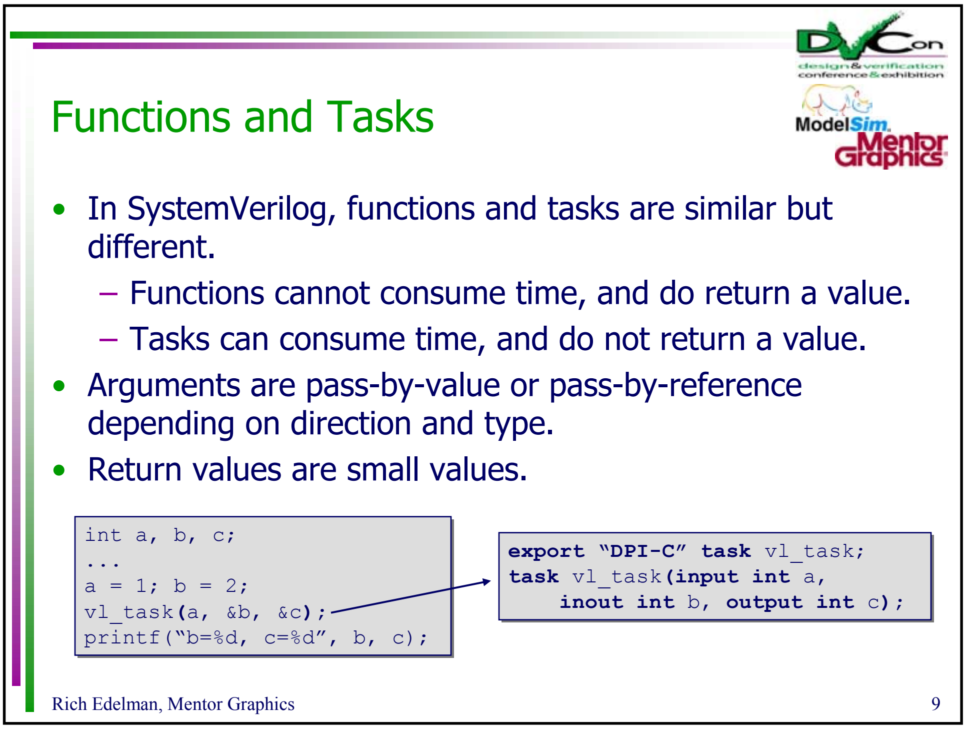
```
void add(int *accumulate, int delta)
{
  *accumulate += delta;
}
```



## Datatypes

- Small values
  - C types
- HW types
  - 2 state
  - 4 state
- Arrays
- Structs
- Strings

| C Data Type | SystemVerilog Data Type |
|---|---|
| char | byte |
| short | shortint |
| int | int |
| long long | longint |
| float | shortreal |
| double | real |
| void * | chandle |
| const char * | string |
| unsigned int | bit |
| unsigned int | logic |

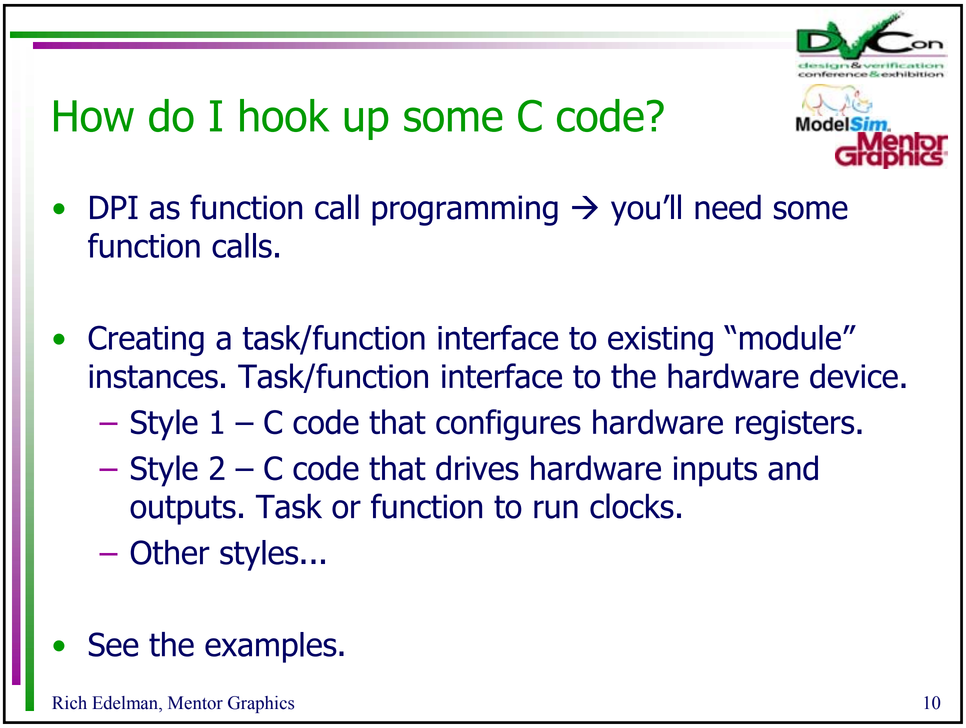


## Functions and Tasks

- In SystemVerilog, functions and tasks are similar but different.
  - Functions cannot consume time, and do return a value.
  - Tasks can consume time, and do not return a value.
- Arguments are pass-by-value or pass-by-reference depending on direction and type.
- Return values are small values.

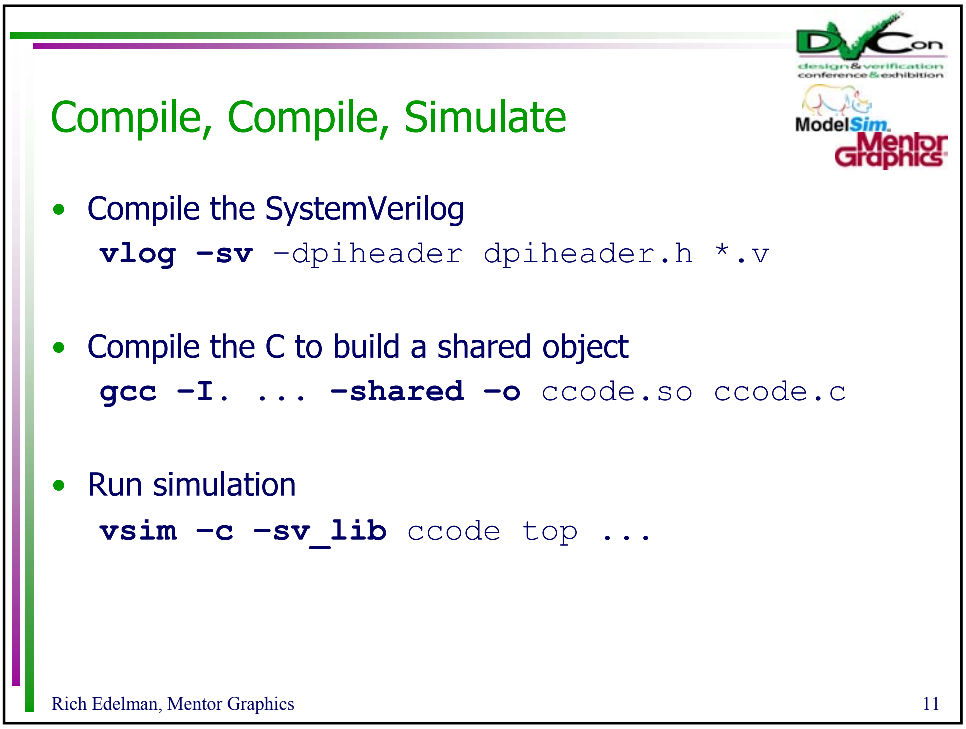
```
int a, b, c;
...
a = 1; b = 2;
vl_task(a, &b, &c);
printf(“b=%d, c=%d”, b, c);
```

```
export “DPI-C” task vl_task;
task vl_task(input int a,
    inout int b, output int c);
```



## How do I hook up some C code?

- DPI as function call programming → you’ll need some function calls.
- Creating a task/function interface to existing “module” instances. Task/function interface to the hardware device.
  - Style 1 – C code that configures hardware registers.
  - Style 2 – C code that drives hardware inputs and outputs. Task or function to run clocks.
  - Other styles...
- See the examples.



## Compile, Compile, Simulate

- Compile the SystemVerilog

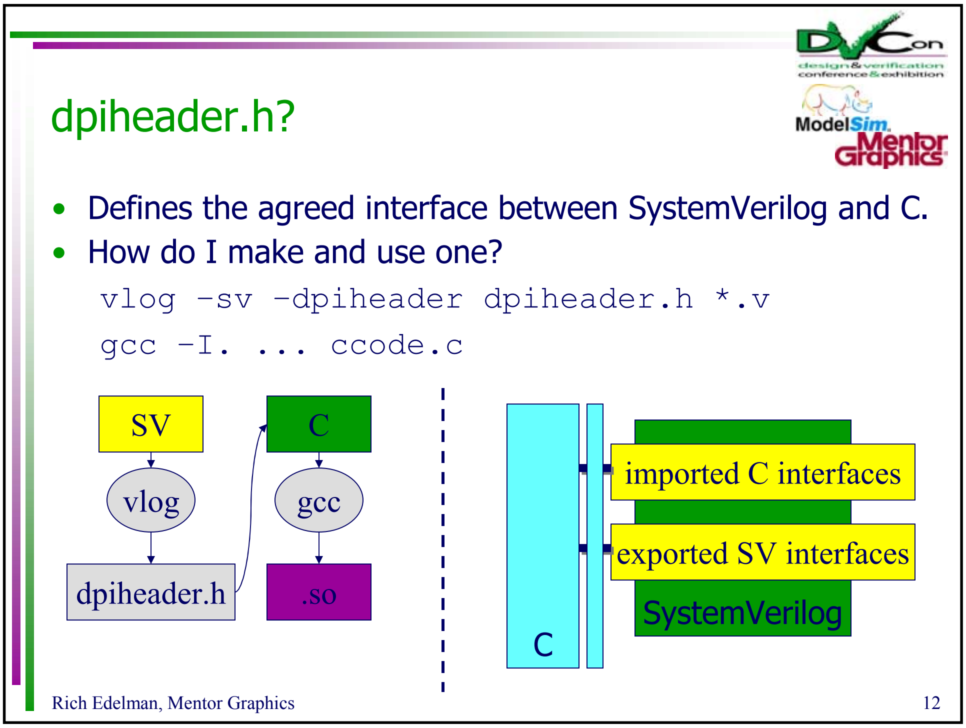
```
vlog -sv -dpiheader dpiheader.h *.v
```

- Compile the C to build a shared object

```
gcc -I. ... -shared -o ccode.so ccode.c
```

- Run simulation

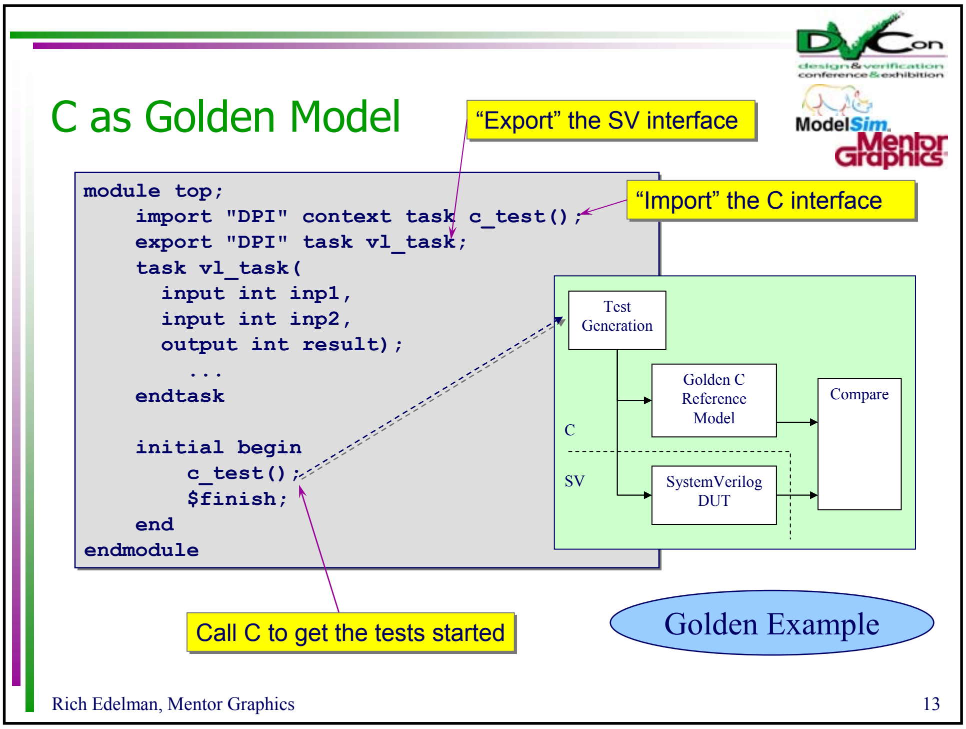
```
vsim -c -sv_lib ccode top ...
```



## dpiheader.h?

- Defines the agreed interface between SystemVerilog and C.
- How do I make and use one?

```
vlog -sv -dpiheader dpiheader.h *.v
gcc -I. ... ccode.c
```



## C as Golden Model

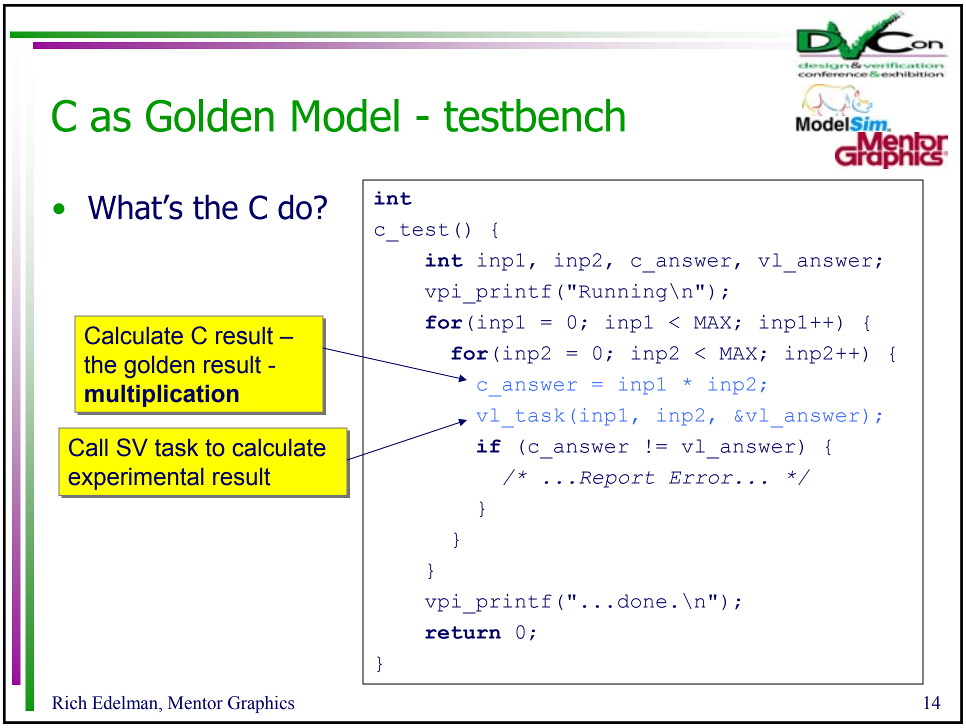
“Export” the SV interface

“Import” the C interface

```
module top;
    import "DPI" context task c_test();
    export "DPI" task vl_task;
    task vl_task(
      input int inp1,
      input int inp2,
      output int result);
        ...
    endtask

    initial begin
        c_test();
        $finish;
    end
endmodule
```

Call C to get the tests started

Golden Example



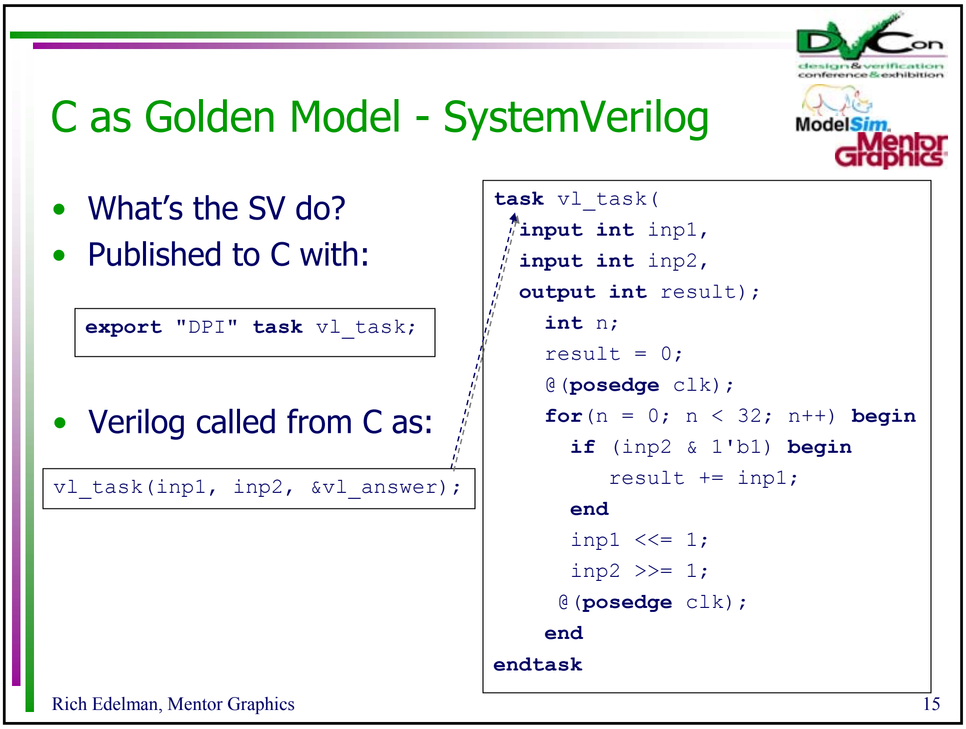
## C as Golden Model - testbench

- What’s the C do?

Calculate C result – the golden result - **multiplication**

Call SV task to calculate experimental result

```
int
c_test() {
    int inp1, inp2, c_answer, vl_answer;
    vpi_printf("Running\n");
    for(inp1 = 0; inp1 < MAX; inp1++) {
      for(inp2 = 0; inp2 < MAX; inp2++) {
        c_answer = inp1 * inp2;
        vl_task(inp1, inp2, &vl_answer);
        if (c_answer != vl_answer) {
          /* ...Report Error... */
        }
      }
    }
    vpi_printf("...done.\n");
    return 0;
}
```



# C as Golden Model - SystemVerilog

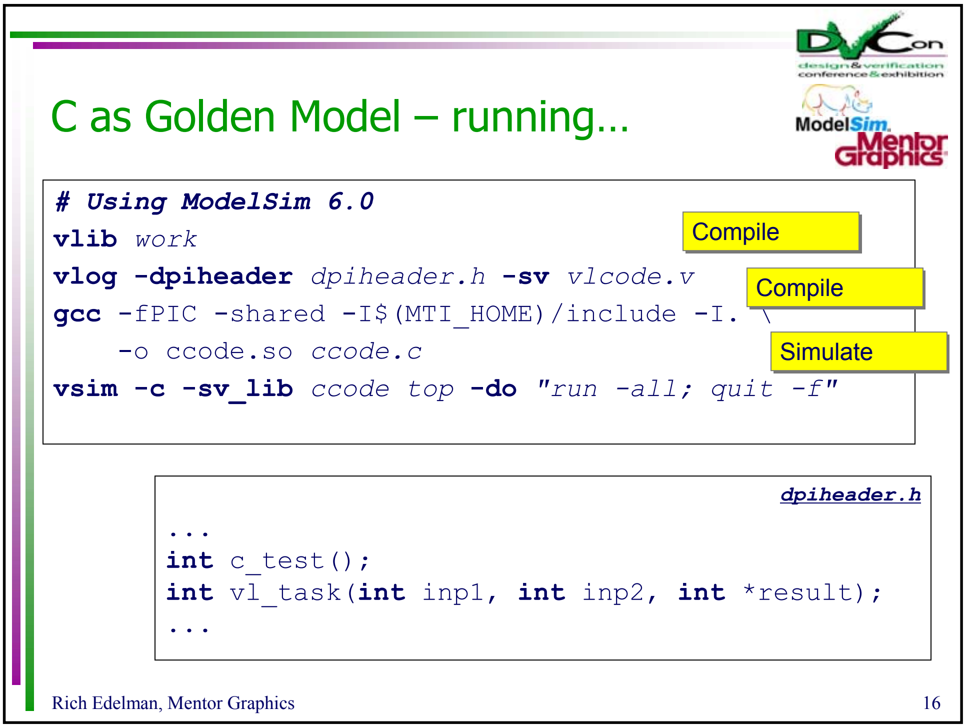
- What’s the SV do?
- Published to C with:

```
export "DPI" task vl_task;
```

- Verilog called from C as:

```
vl_task(inp1, inp2, &vl_answer);
```

```
task vl_task(
  input int inp1,
  input int inp2,
  output int result);
    int n;
    result = 0;
    @(posedge clk);
    for(n = 0; n < 32; n++) begin
      if (inp2 & 1'b1) begin
         result += inp1;
      end
      inp1 <<= 1;
      inp2 >>= 1;
     @(posedge clk);
    end
endtask
```



# C as Golden Model – running…

```
# Using ModelSim 6.0
vlib work
vlog -dpiheader dpiheader.h -sv vlcode.v
gcc -fPIC -shared -I$(MTI_HOME)/include -I. \
     -o ccode.so ccode.c
vsim -c -sv_lib ccode top -do "run -all; quit -f"
```

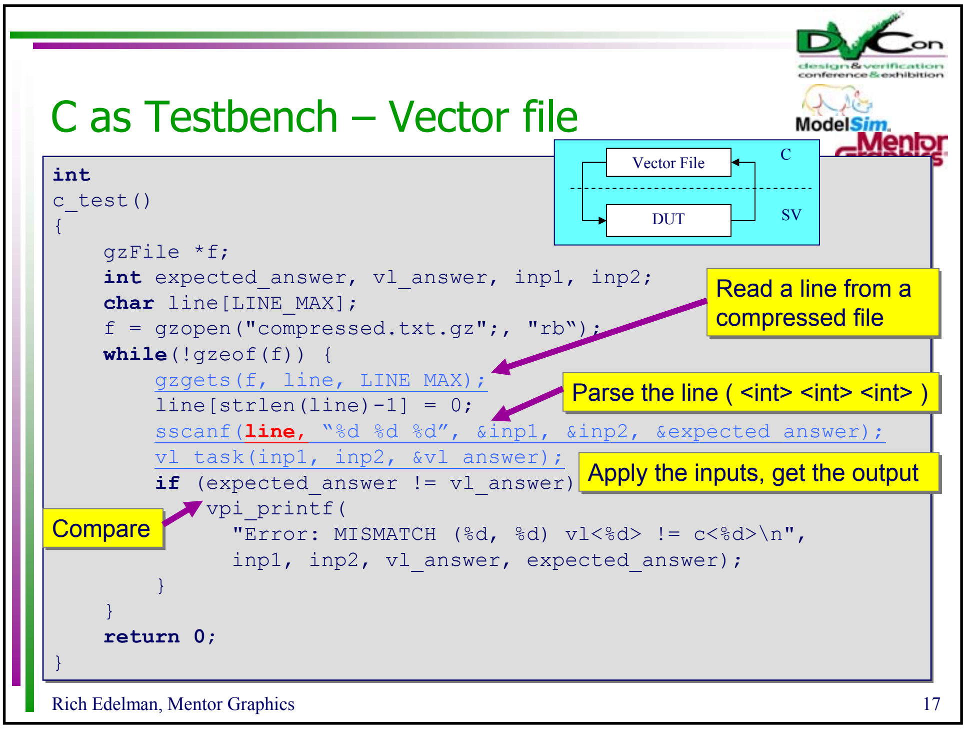
Compile

Compile

Simulate

dpiheader.h

```
...
int c_test();
int vl_task(int inp1, int inp2, int *result);
...
```



## C as Testbench – Vector file

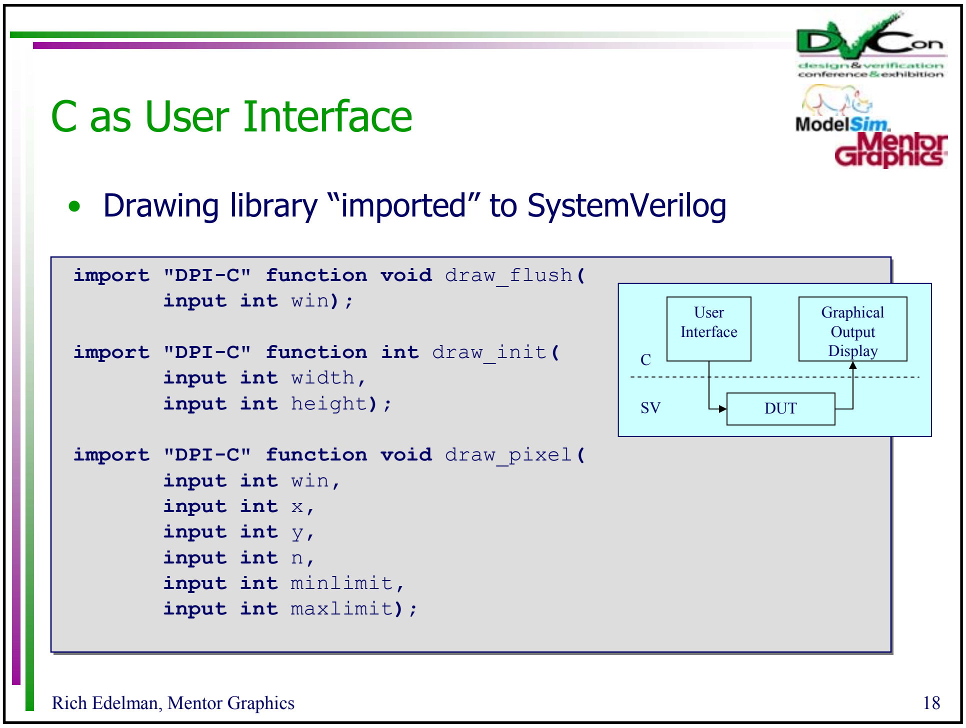
```
int
c_test()
{
    gzFile *f;
    int expected_answer, vl_answer, inp1, inp2;
    char line[LINE_MAX];
    f = gzopen("compressed.txt.gz";, "rb");
    while(!gzeof(f)) {
        gzgets(f, line, LINE_MAX);
        line[strlen(line)-1] = 0;
        sscanf(line, “%d %d %d”, &inp1, &inp2, &expected_answer);
        vl_task(inp1, inp2, &vl_answer);
        if (expected_answer != vl_answer)
            vpi_printf(
              "Error: MISMATCH (%d, %d) vl<%d> != c<%d>\n",
              inp1, inp2, vl_answer, expected_answer);
        }
    }
    return 0;
}
```

Read a line from a compressed file

Parse the line ( <int> <int> <int> )

Apply the inputs, get the output

Compare

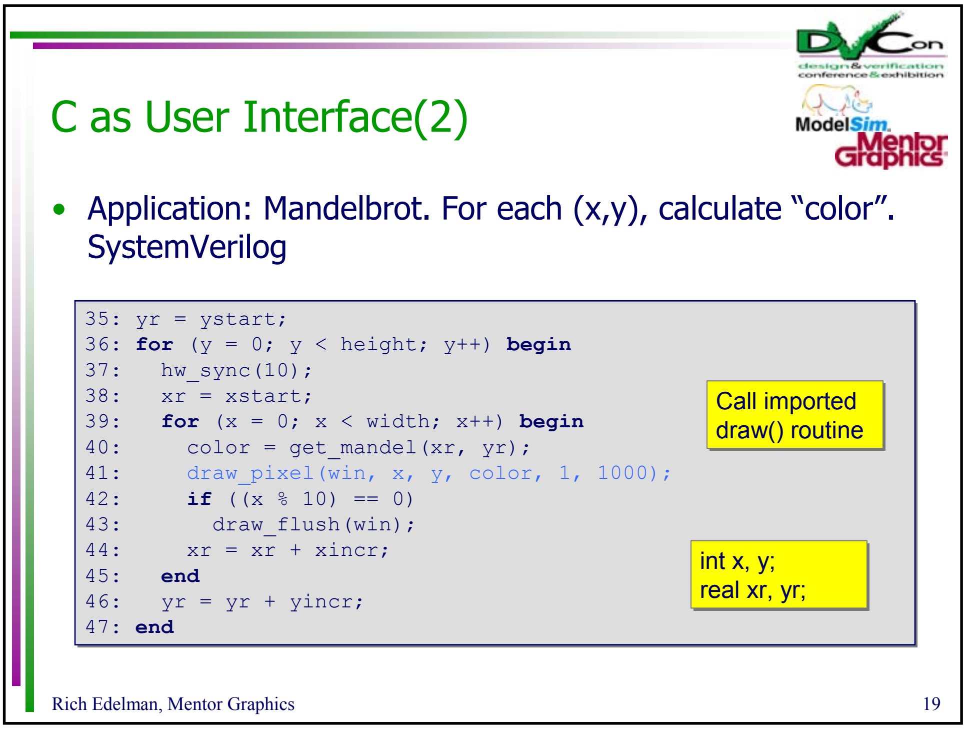


## C as User Interface

- Drawing library “imported” to SystemVerilog

```
import "DPI-C" function void draw_flush(
       input int win);

import "DPI-C" function int draw_init(
       input int width,
       input int height);

import "DPI-C" function void draw_pixel(
       input int win,
       input int x,
       input int y,
       input int n,
       input int minlimit,
       input int maxlimit);
```

User Interface
Graphical Output Display
C
SV
DUT



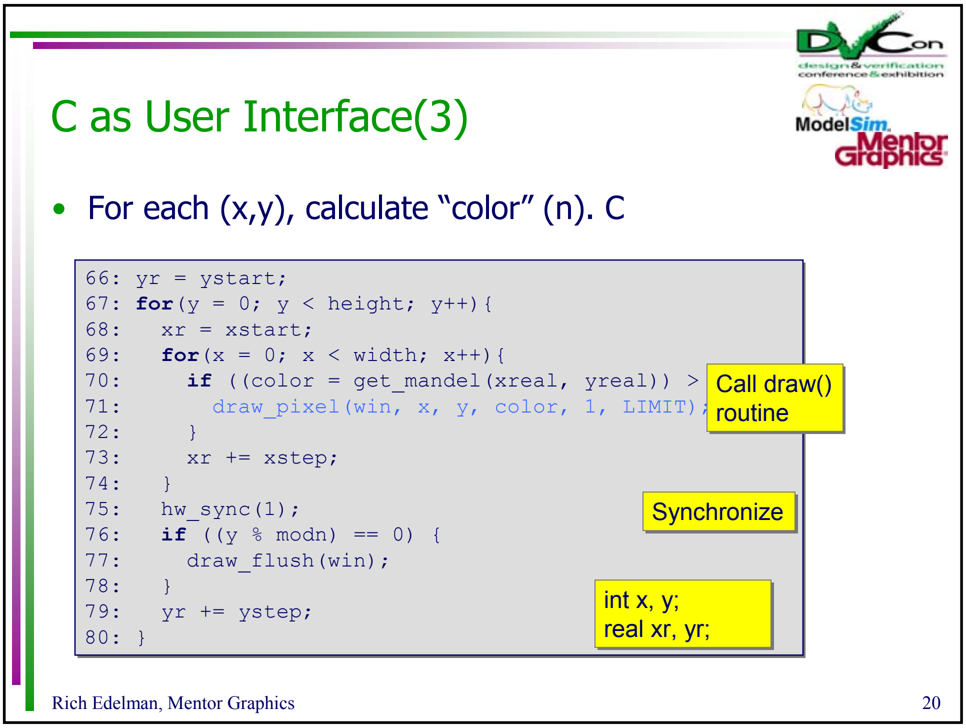
## C as User Interface(2)

- Application: Mandelbrot. For each (x,y), calculate “color”. SystemVerilog

```
yr = ystart;
for (y = 0; y < height; y++) begin
  hw_sync(10);
  xr = xstart;
  for (x = 0; x < width; x++) begin
    color = get_mandel(xr, yr);
    draw_pixel(win, x, y, color, 1, 1000);
    if ((x % 10) == 0)
      draw_flush(win);
    xr = xr + xincr;
  end
  yr = yr + yincr;
end
```

Call imported draw() routine

int x, y;
real xr, yr;



## C as User Interface(3)

- For each (x,y), calculate “color” (n). C

```
yr = ystart;
for(y = 0; y < height; y++){
  xr = xstart;
  for(x = 0; x < width; x++){
    if ((color = get_mandel(xreal, yreal)) >
      draw_pixel(win, x, y, color, 1, LIMIT);
    }
    xr += xstep;
  }
  hw_sync(1);
  if ((y % modn) == 0) {
    draw_flush(win);
  }
  yr += ystep;
}
```

Call draw() routine

Synchronize

int x, y;
real xr, yr;

## C as User Interface(4)

- hw_sync() – sharing the CPU

```
export "DPI-C" task hw_sync;
task automatic hw_sync(input int count);
  while(count-- > 0) begin @(posedge sync_clk); end
endtask

initial begin
fork
 begin vl_mandel(200, 200, 0.075, 0.175, 0.59, 0.69); end
 begin  c_mandel(200, 200, 0.075, 0.175, 0.59, 0.69); end
 begin  c_mandel(100, 100, 0.075, 0.175, 0.59, 0.69); end
join
$finish;
end

always begin sync_clk = 0; #1; sync_clk = 1; #1; end
```

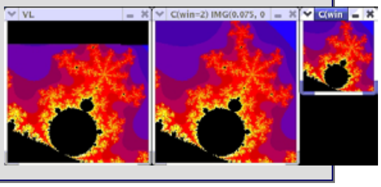
Synchronizer

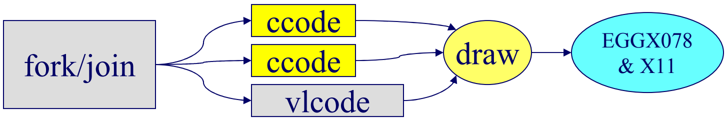
Threads

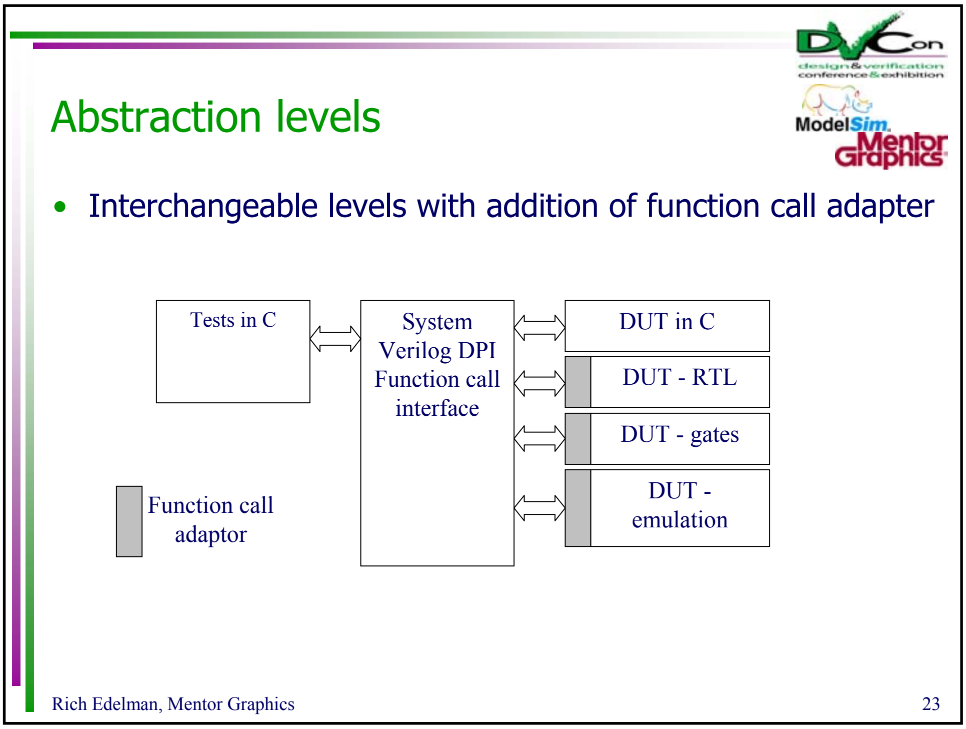
## C as User Interface(5)

- Compile, Compile, Simulate...(and load libraries)

```
vlog -dpiheader dpiheader.h -sv vlcode.v

gcc -fPIC -shared $(CFLAGS) \
   -Idraw -I$(MTI_HOME)/include -I. \
   -o ccode.so \
   -L./draw -L/usr/X11R6/lib \
   -ldraw -lX11 ccode.c

vsim top -sv_lib ccode -c \
   -do "run -all; quit -f"
```

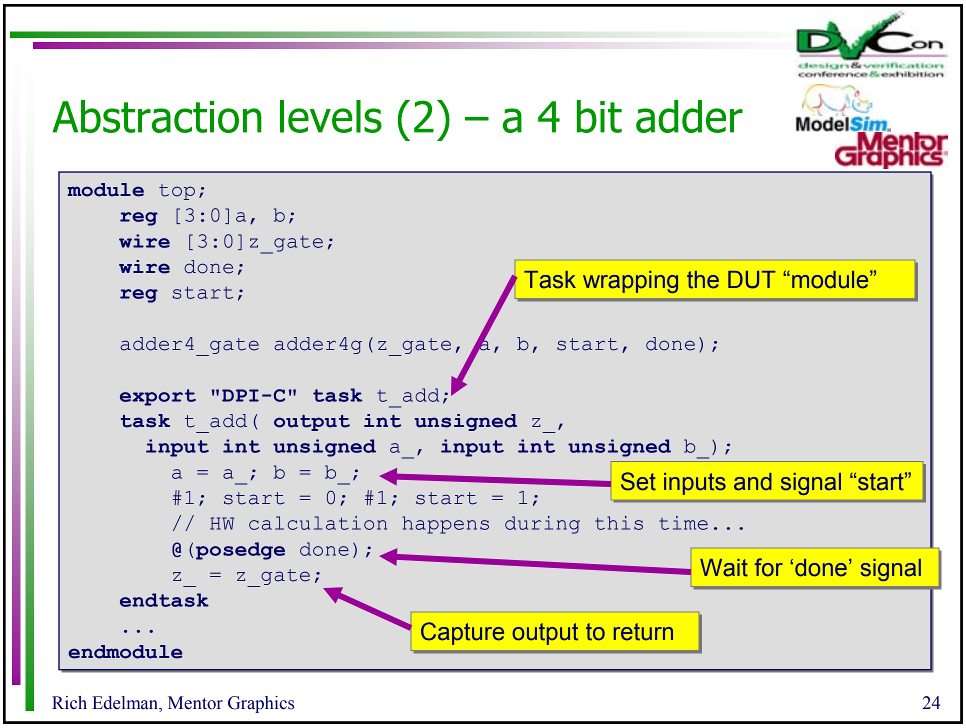






## Abstraction levels

- Interchangeable levels with addition of function call adapter



## Abstraction levels (2) – a 4 bit adder

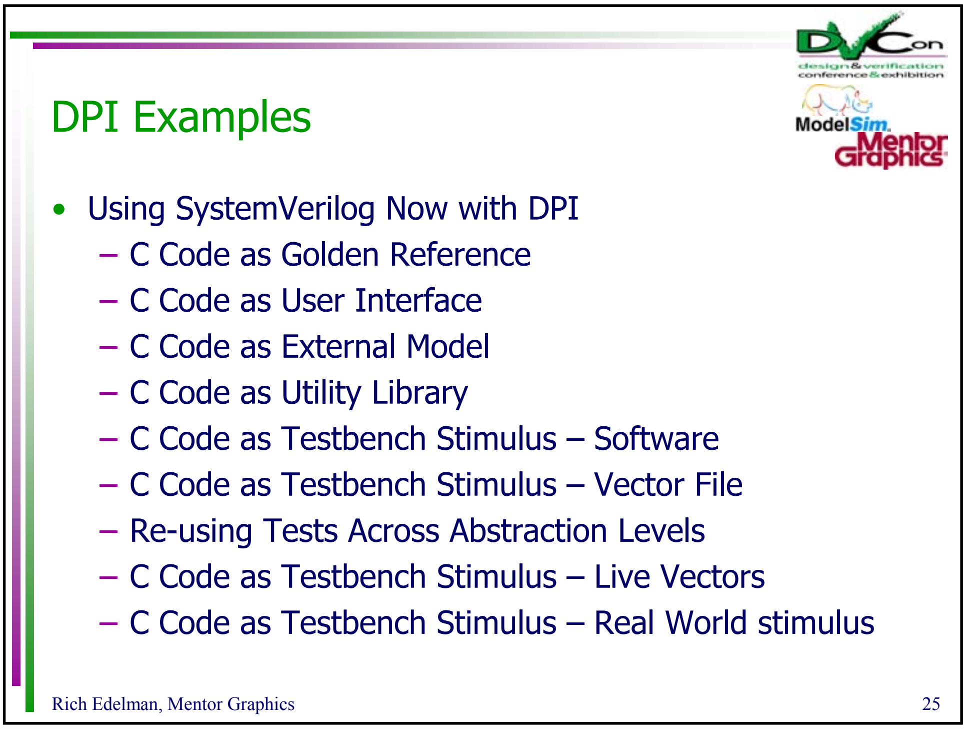
```
module top;
    reg [3:0]a, b;
    wire [3:0]z_gate;
    wire done;
    reg start;

    adder4_gate adder4g(z_gate, a, b, start, done);

    export "DPI-C" task t_add;
    task t_add( output int unsigned z_,
      input int unsigned a_, input int unsigned b_);
        a = a_; b = b_;
        #1; start = 0; #1; start = 1;
        // HW calculation happens during this time...
        @(posedge done);
        z_ = z_gate;
    endtask
    ...
endmodule
```

Task wrapping the DUT “module”

Set inputs and signal “start”

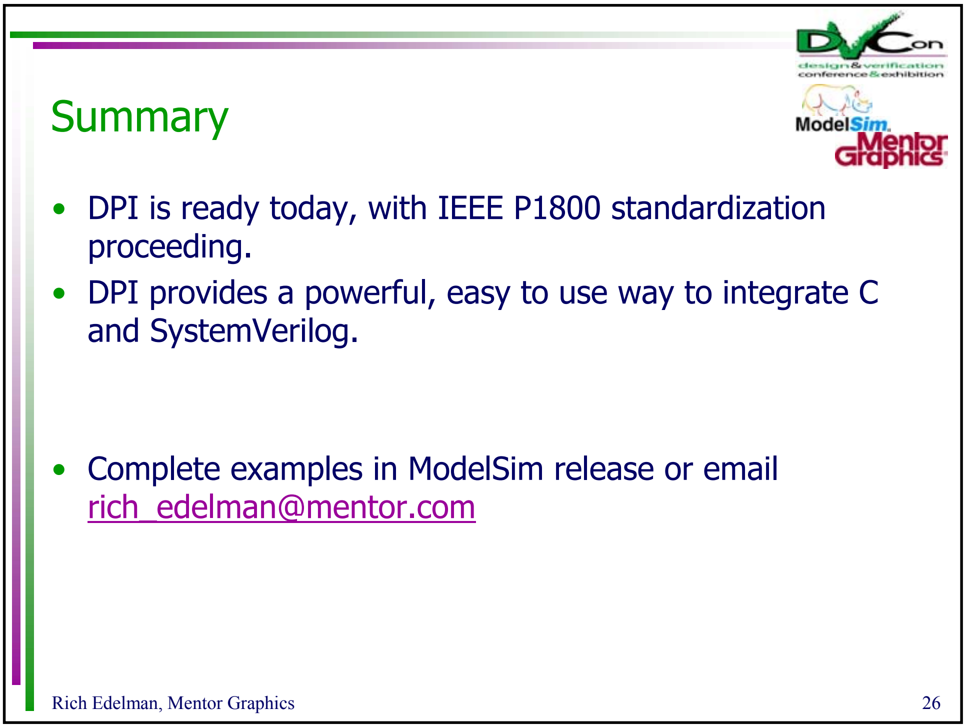
Wait for ‘done’ signal

Capture output to return



## DPI Examples

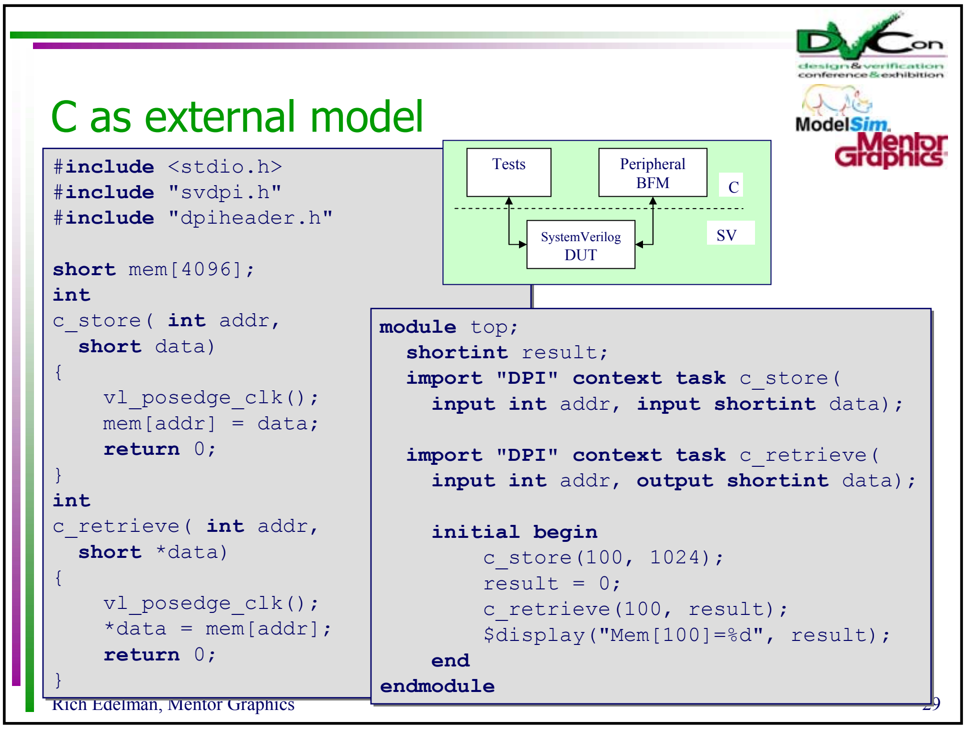
- Using SystemVerilog Now with DPI
  - C Code as Golden Reference
  - C Code as User Interface
  - C Code as External Model
  - C Code as Utility Library
  - C Code as Testbench Stimulus – Software
  - C Code as Testbench Stimulus – Vector File
  - Re-using Tests Across Abstraction Levels
  - C Code as Testbench Stimulus – Live Vectors
  - C Code as Testbench Stimulus – Real World stimulus



## Summary

- DPI is ready today, with IEEE P1800 standardization proceeding.
- DPI provides a powerful, easy to use way to integrate C and SystemVerilog.
- Complete examples in ModelSim release or email rich_edelman@mentor.com

## Bonus Slides



# C as external model

Tests

Peripheral BFM

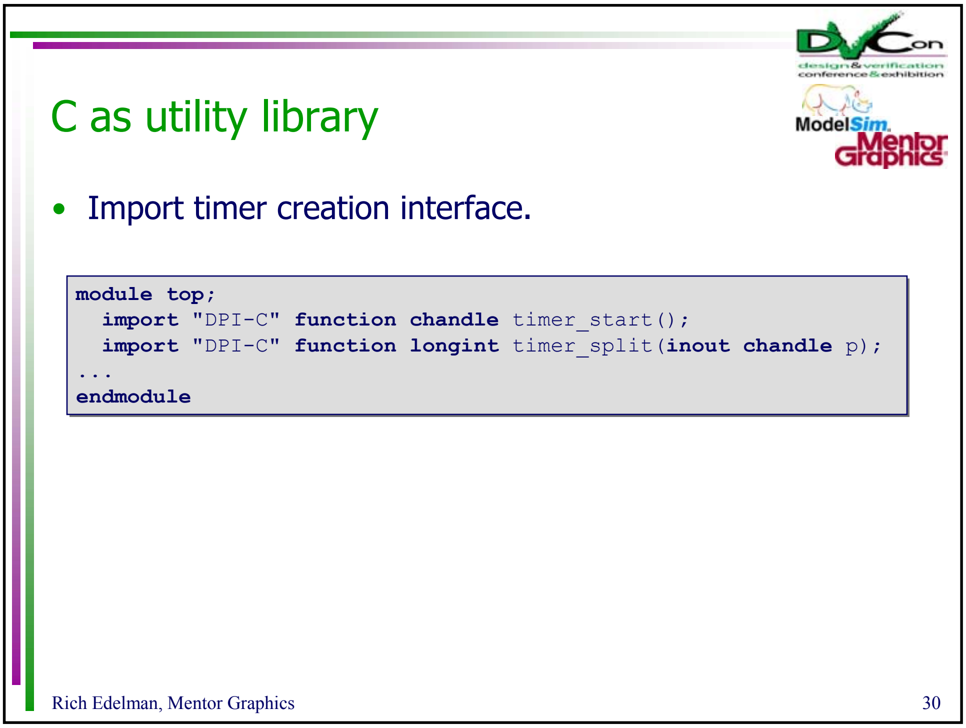
C

SystemVerilog DUT

SV

```
#include <stdio.h>
#include "svdpi.h"
#include "dpiheader.h"

short mem[4096];
int
c_store( int addr,
  short data)
{
    vl_posedge_clk();
    mem[addr] = data;
    return 0;
}
int
c_retrieve( int addr,
  short *data)
{
    vl_posedge_clk();
    *data = mem[addr];
    return 0;
}
```

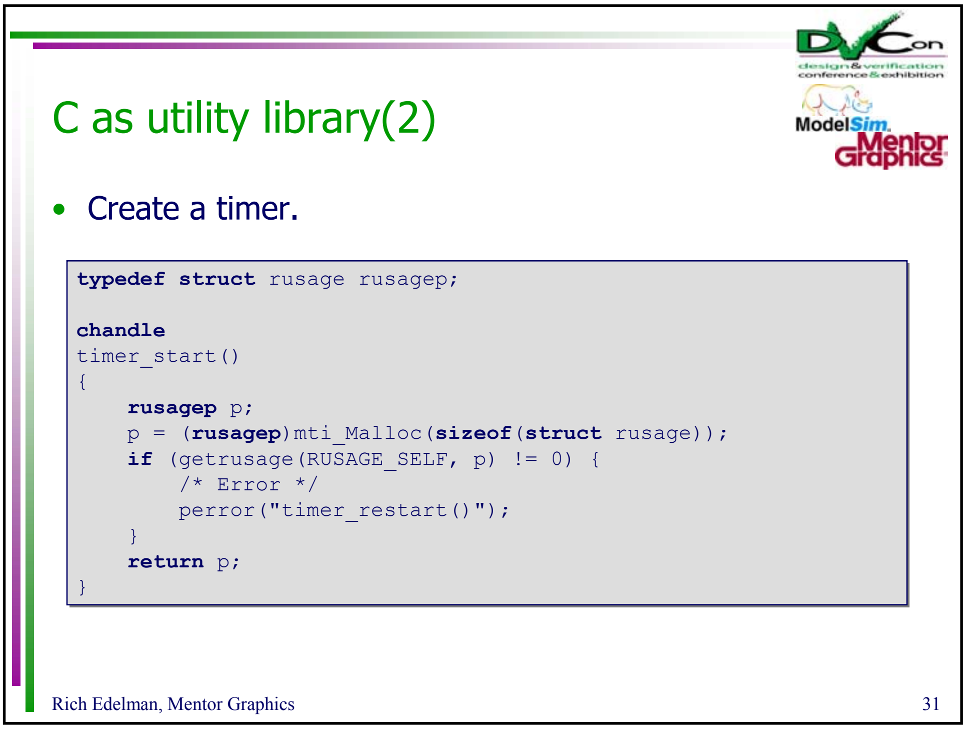
```
module top;
  shortint result;
  import "DPI" context task c_store(
    input int addr, input shortint data);

  import "DPI" context task c_retrieve(
    input int addr, output shortint data);

    initial begin
        c_store(100, 1024);
        result = 0;
        c_retrieve(100, result);
        $display("Mem[100]=%d", result);
    end
endmodule
```



# C as utility library

- Import timer creation interface.

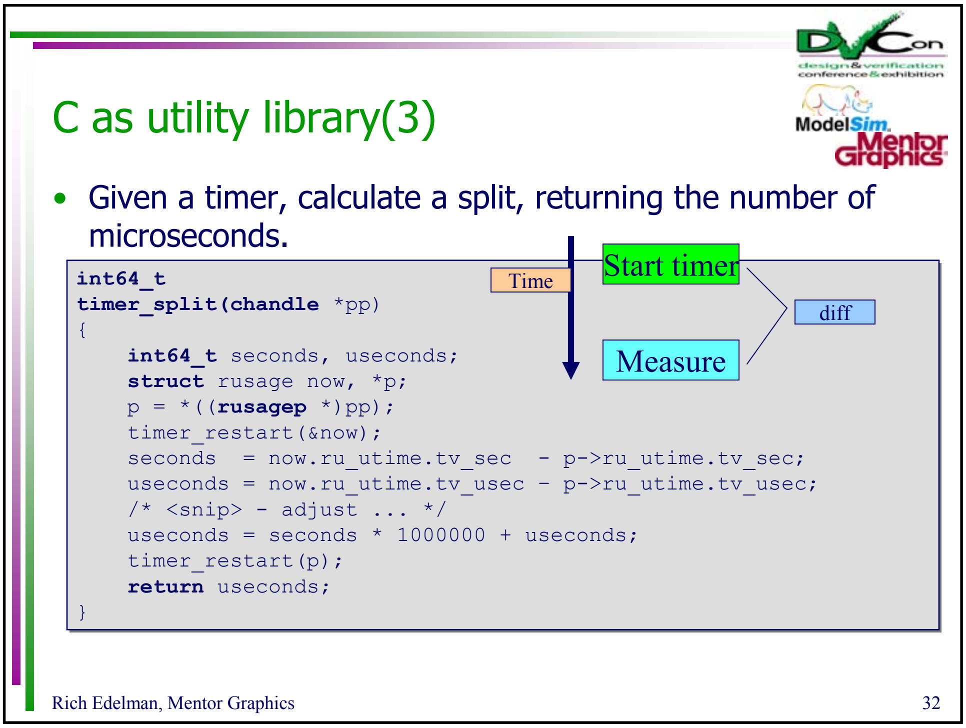
```
module top;
  import "DPI-C" function chandle timer_start();
  import "DPI-C" function longint timer_split(inout chandle p);
...
endmodule
```



## C as utility library(2)

- Create a timer.

```
typedef struct rusage rusagep;

chandle
timer_start()
{
    rusagep p;
    p = (rusagep)mti_Malloc(sizeof(struct rusage));
    if (getrusage(RUSAGE_SELF, p) != 0) {
        /* Error */
        perror("timer_restart()");
    }
    return p;
}
```



## C as utility library(3)

- Given a timer, calculate a split, returning the number of microseconds.

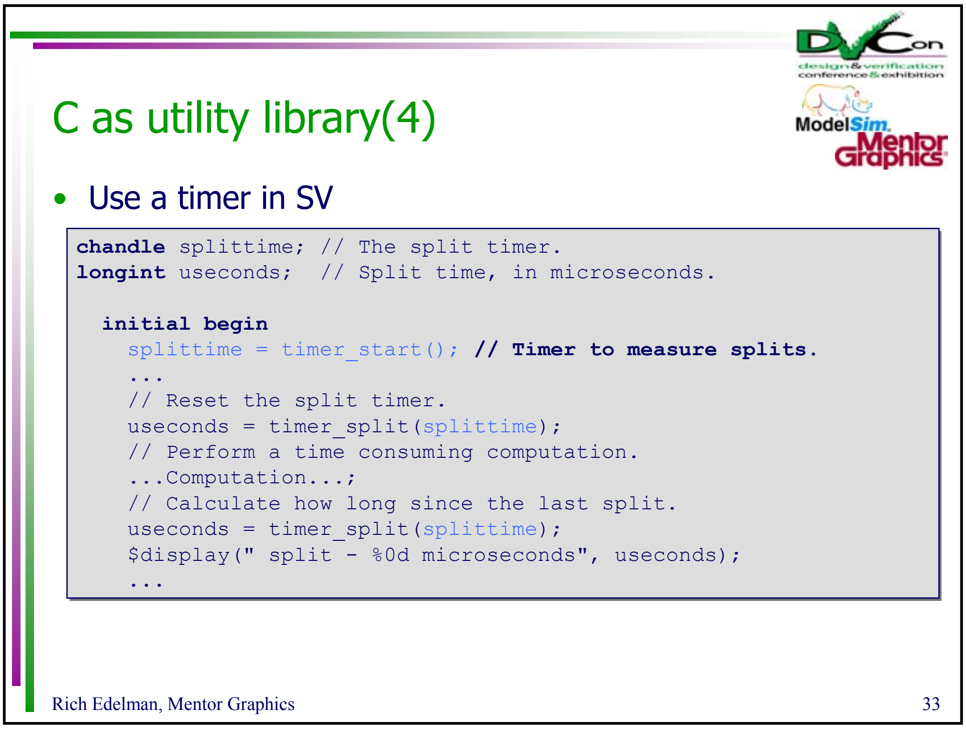
Time

Start timer

Measure

diff

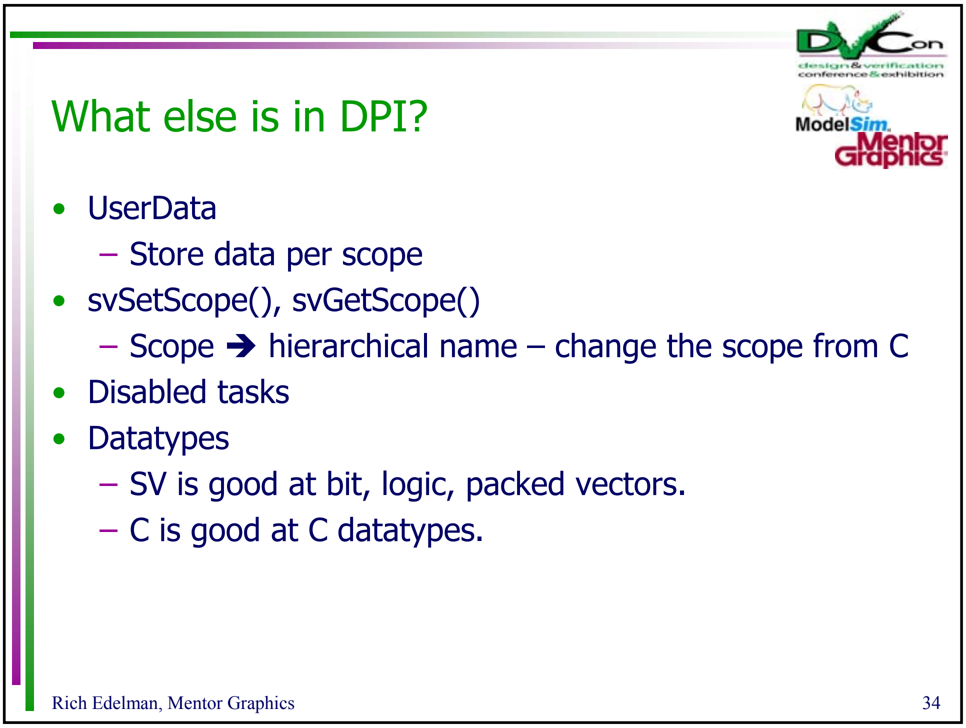
```
int64_t
timer_split(chandle *pp)
{
    int64_t seconds, useconds;
    struct rusage now, *p;
    p = *((rusagep *)pp);
    timer_restart(&now);
    seconds  = now.ru_utime.tv_sec  - p->ru_utime.tv_sec;
    useconds = now.ru_utime.tv_usec - p->ru_utime.tv_usec;
    /* <snip> - adjust ... */
    useconds = seconds * 1000000 + useconds;
    timer_restart(p);
    return useconds;
}
```



## C as utility library(4)

- Use a timer in SV

```
chandle splittime; // The split timer.
longint useconds;  // Split time, in microseconds.

  initial begin
    splittime = timer_start(); // Timer to measure splits.
    ...
    // Reset the split timer.
    useconds = timer_split(splittime);
    // Perform a time consuming computation.
    ...Computation...;
    // Calculate how long since the last split.
    useconds = timer_split(splittime);
    $display(" split - %0d microseconds", useconds);
    ...
```



## What else is in DPI?

- UserData
  - Store data per scope
- svSetScope(), svGetScope()
  - Scope ➔ hierarchical name – change the scope from C
- Disabled tasks
- Datatypes
  - SV is good at bit, logic, packed vectors.
  - C is good at C datatypes.

## Careful

- When C code is started, it runs until
  - It returns to SystemVerilog, *or*
  - It calls a SystemVerilog task that consumes time.
- Begin in SV.
- Threaded programming can be hard
- Imported C tasks are defined to return 'int'. Return 0 for normal exit. Return 1 if the function was 'disabled'.
- Memory ownership: caller owns the memory.
- Calling `sleep(n)` from C ...