
Rich Edelman, Rohit Jain, Hui Yin
Mentor Graphics, Fremont, CA
SystemVerilog DPI-C

- Call **C code** from SystemVerilog “natively”
  - Just like calling SystemVerilog code

- Call **SystemVerilog code** from C “natively”
  - Nothing special, just push the arguments on the stack and jump

- That’s it. Function calls…

<table>
<thead>
<tr>
<th>C</th>
<th>SystemVerilog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function Call</td>
<td>Function Call</td>
</tr>
<tr>
<td>Task Call</td>
<td></td>
</tr>
</tbody>
</table>
But, ....there is an API

- If you are **HEAVILY** using it, check yourself.
- If you are **fiddling around with data layout**, check.

→ Let SV do the bit manipulations
→ Let C do the ‘int’ things

```c
typedef struct t_vpi_vecval {
    uint32_t aval;
    uint32_t bval;
} s_vpi_vecval, *p_vpi_vecval;

typedef s_vpi_vecval svLogicVecVal;
typedef void*        svOpenArrayHandle;

void svPutLogicArrElem3VecVal(const svOpenArrayHandle d,  
                              const svLogicVecVal* s, int indx1, int indx2, int indx3);
void svGetLogicArrElem3VecVal(svLogicVecVal* d,  
                              const svOpenArrayHandle s, int indx1, int indx2, int indx3);
```

Paper has
- Example code to explore **data layout**
- Example code listing the API

Keep it Simple
• What’s the difference between a task and a function?
  – Tasks can consume time. Functions cannot.
  – Functions have a return value. Tasks do not.
• What’s does it mean for C code to consume time?

C Code
int

c_task() {
    wait_n_clocks(10);
    return 0;
}

SV Code

task wait_n_clocks(int n);
    repeat(n)@(posedge clk);
endtask
Imports and Exports

**C Code – IMPORTED CODE**

```c
void c_function(...) {
    sv_task(...);
    sv_functionX(...);
}

int c_task(...) {
    sv_taskX(...);
    sv_function(...);
}
```

**SV Code – EXPORTED CODE**

```sv
export "DPI-C" task sv_task;
export "DPI-C" function sv_function;
import "DPI-C" task c_task(...);
import "DPI-C" function void c_function(...);

task sv_task(...);
    c_function(...);

endtask

function void sv_function(...);
    c_taskX(...);
    c_function(...);
endfunction
```

Can’t call a task from a function
Inputs, Outputs and Inouts

**SV Code**

```c
import "DPI-C" function void c_adder(
    input int a, input int b,
    output int c, inout int d);

function void sv_function();
    int x, y, z, w;
    x = 1; y = 2; w = 100;
    c_adder(x, y, z, w);
endfunction
```

```c
void c_adder(
    int a, int b,
    int* c, int* d)
{
    *c = a + b;
    *d = *d + *c;
}
```
Passing Data
Copy versus Reference

Copy of Giant Array of Data 10MB

C Code
Copy 10M bytes

SV Code
Reference to Copy 10M bytes

Copy 4 bytes

Giant Array of Data 10MB
Example: Array of integers...

```c
import "DPI-C" function int f_int_openarray (  
    input int i_array[],  
    output int o_array[]);  

ret_bit = f_int_openarray(i_openarray, o_openarray);

int f_int_openarray(  
    const svOpenArrayHandle i_array,  
    const svOpenArrayHandle o_array) {  
    int j, n;  
    int *i, *o;  
    n = svSize(i_array, 1);  
    i = svGetArrayPtr(i_array);  
    o = svGetArrayPtr(o_array);  
    for (j = 0; j < n; j++) {  
        o[j] = i[j] + 1;  
    }  
    return 0;
}
```
Example: AES Encryption

- Build the AES shared library – out-of-the-box

  ```
  Makefile
  gcc -Wall -Os -Wl,-Map,test.map -c aes.c -o aes.o -fPIC
  gcc -o aes.so aes.o -shared
  ```

- Generate a header and link the shared library

  ```
  Makefile
  vlib work
  vlog -dpiheader dpiheader.h t.sv
  vopt -o opt top
  vsim -c opt ... -sv_lib ../tiny-AES128-C-master/aes
  ```

  [https://github.com/kokke/tiny-AES128-C](https://github.com/kokke/tiny-AES128-C)
C Code – NO change from the Internet

```c
void AES128_CBC_encrypt_buffer(uint8_t* output,
                                uint8_t* input,
                                uint32_t length,
                                const uint8_t* key,
                                const uint8_t* iv)
{
    ... 
    for (i = 0; i < length; i += KEYLEN)
    {
        XorWithIv(input);
        BlockCopy(output, input);
        state = (state_t*)output;
        Cipher();
        Iv = output;
        input += KEYLEN;
        output += KEYLEN;
    }
    ... 
}
```

SV Code

```c
typedef byte unsigned uint8_t;
import "DPI-C" function void AES128_CBC_encrypt_buffer(
    output uint8_t buffer[64],
    input uint8_t in[64],
    input int size,
    input uint8_t key[16],
    input uint8_t iv[16]);

dpiheader.h
void AES128_CBC_encrypt_buffer(
    unsigned char* buffer,
    const unsigned char* in,
    int size,
    const unsigned char* key,
    const unsigned char* iv);
```
module top();

typedef byte unsigned uint8_t;

uint8_t key[] = '{ 8'h2b, 8'h7e, ... 8'h4f, 8'h3c };
uint8_t iv[]  = '{ 8'h00, 8'h01, ... 8'h0e, 8'h0f };
uint8_t in[]  = '{ 8'h6b, 8'hc1, ... 8'h17, 8'h2a, 8'hae, 8'h2d, ... 8'h8e, 8'h51, 8'h30, 8'hc8, ... 8'h52, 8'hef, 8'hf6, 8'h9f, ... 8'h37, 8'h10 };
uint8_t out[] = '{ 8'h76, 8'h49, ... 8'h19, 8'h7d, 8'h50, 8'h86, ... 8'h78, 8'hb2, 8'h73, 8'hbe, ... 8'h95, 8'h16, 8'h3f, 8'hf1, ... 8'he1, 8'ha7 };
uint8_t buffer[64];
SV Code

```sv
import "DPI-C" function void AES128_CBC_encrypt_buffer(
    output uint8_t buffer[64],
    input uint8_t in[64],
    input int size,
    input uint8_t key[16],
    input uint8_t iv[16]);

initial begin
  AES128_CBC_encrypt_buffer(buffer, in, 64, key, iv);

  if (buffer != out)
    $display("FATAL: Mismatch");
  else
    $display("INFO: Pass");
end
endmodule
```

Call DPI-C code

Compare
Building a UVM TB with C Golden Model

```vhdl
task body();

GoldenAES(expected, nist_aes128_key, ibv);

wseq.start(mem_sqr);

cseq.transaction_type = WRITE;
cseq.word = wseq.addr;
cseq.start(cfg_sqr);

cseq.addr = i+1;
cseq.word = 1; // The GO signal
cseq.start(cfg_sqr);

cseq.transaction_type = WAIT_FOR_INTERRUPT;
cseq.start(cfg_sqr);

rseq.start(mem_sqr);

obv = ...

if (obv !== expected)
  `uvm_fatal(get_type_name(), "Fatal mismatch")
else
  `uvm_info(get_type_name(), "Match", UVM_MEDIUM)
endtask
```

- Call the GOLDEN model
- Write the input data to memory
- Configure
- Wait for interrupt – AES done
- Read the output from memory
- Compare
What about Threading?

- **Easy**
  - 1 SV Thread
  - (Not really threaded)

- **Recipe**
  - 1 SV Thread
  - C Threads start and stop

- **Expert**
  - 1 SV Thread
  - C Threads never end
  - (Can be quite tricky)
Many Fractal Threads
SV & C code

SV Code
module top();
import "DPI-C" context task c_mandel(
    input int xpos, ypos, width, height,
    real xstart, xend, ystart, yend);

initial begin
    fork
        m1.vl_mandel(0*(X), 0, 800, 800, xstart, xend, ystart, yend);
        c_mandel(1*(X), 0, 800, 800, xstart, xend, ystart, yend);
        c_mandel(2*(X), 0*(Y), 300, 300, xstart, xend, ystart, yend);
        m2.vl_mandel(2*(X), 1*(Y), 300, 300, xstart, xend, ystart, yend);
        m3.vl_mandel(2*(X), 2*(Y), 300, 300, xstart, xend, ystart, yend);
        c_mandel(2*(X), 3*(Y), 100, 100, xstart, xend, ystart, yend);
    join
    $finish;
end
endmodule

“6” SV Threads
3 Verilog
3 C
SV Code

```sv
task vl_mandel(input int xpos, input int ypos, input int width, input int height, input real xstart, input real xend, input real ystart, input real yend);

string label;

win = draw_init(xpos, ypos, width, height);
label = $sformatf("VL(win=%0d) IMG(%g, %g) (%g, %g)", win, xstart, ystart, xend, yend);

draw_title(win, label);

xstep = (xend - xstart)/width;
ystep = (yend - ystart)/height;

yr = ystart;
for (y = 0; y < height; y++) begin
    hw_sync(1);
    draw_flush(win);
    xr = xstart;
    for (x = 0; x < width; x++) begin
        n = get_mandel(xr, yr);
        draw_pixel(win, x, y, n, 1, 1000);
        xr += xstep;
    end
    yr += ystep;
end

draw_finish(win);
draw_clear(win);
endtask
```

SV Code

```sv
task vl_mandel(...);
    for (y = 0; y < height; y++) begin
        hw_sync(1);

        for (x = 0; x < width; x++) begin
            n = get_mandel(xr, yr);
            draw_pixel(win, x, y, n, 1, 1000);
        end
    end
endtask
```
C Code

```c
int c_mandel(...)
{
    int n, x, y;
    int win;
    char label[1024];
    win = draw_init(xpos, ypos, width, height);
    sprintf(label, " C(win=%d) IMG(%g, %g) (%g, %g)",
            win, xstart, ystart, xend, yend);
    draw_title(win, label);
    xstep = (xend - xstart)/width;
    ystep = (yend - ystart)/height;
    yr = ystart;
    for(y = 0; y < height; y++) {
        hw_sync(1);
        draw_flush(win);
        xr = xstart;
        for(x = 0; x < width; x++) {
            n = get_mandel(xr, yr);
            draw_pixel(win, x, y, n, 1, LIMIT);
            xr += xstep;
        }
        yr += ystep;
    }
    draw_finish(win);
    draw_clear(win);
    return 0;
}
```

Y Pixels

X Pixels

Calculate

Draw
hw_sync() – taking turns

SV Code

export "DPI-C" task hw_sync;

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

task automatic hw_sync(input int count);
    repeat(count)
        @(posedge sync_clk);
endtask
The C code also draws dots.
  – Calling the same API in C.

SV Code

```c
import "DPI-C" function void draw_finish (input int win);
import "DPI-C" function void draw_flush (input int win);
import "DPI-C" function void draw_clear (input int win);
import "DPI-C" function void draw_title (input int win, string title);
import "DPI-C" function int draw_init (input int xpos, ypos, width, height);
import "DPI-C" function void draw_pixel (input int win, x, y, n, minlimit, maxlimit);
```
Lots of Dots

### Inner Loop: Calculate What Color My Pixel Should Be

```function int get_mandel(input real c_real, input real c_imaginary);
    re = 0;
    im = 0;
    for (i = 0; i < `LIMIT; i++) begin
        if (((xsq = (re * re)) + (ysq = (im * im))) > `THRESHOLD)
            break;
        tmp = xsq - ysq + c_real;
        im = 2.0 * re * im + c_imaginary;
        re = tmp;
    end
    if (i == `LIMIT) return 1;
    else return i;
endfunction```

Total in 6 images:

- **318M inner loop evaluations**
- **1.56M dots drawn**

<table>
<thead>
<tr>
<th>Window</th>
<th>IMG Width</th>
<th>IMG Height</th>
<th>Total Dots</th>
<th>Inner Loop Evaluations</th>
</tr>
</thead>
<tbody>
<tr>
<td>VL</td>
<td>0.075</td>
<td>0.59</td>
<td>640,000</td>
<td>130,241,256</td>
</tr>
<tr>
<td>C</td>
<td>0.075</td>
<td>0.59</td>
<td>640,000</td>
<td>130,241,256</td>
</tr>
<tr>
<td>C</td>
<td>0.075</td>
<td>0.59</td>
<td>90,000</td>
<td>18,378,667</td>
</tr>
<tr>
<td>VL</td>
<td>0.075</td>
<td>0.59</td>
<td>90,000</td>
<td>18,378,667</td>
</tr>
<tr>
<td>VL</td>
<td>0.075</td>
<td>0.59</td>
<td>10,000</td>
<td>2,072,564</td>
</tr>
<tr>
<td>C</td>
<td>0.075</td>
<td>0.59</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Parallel and Serial Drawing
Prime Number Algorithm

• Finding a prime is independent of the other primes
  – Perfect for easy parallelization

• Find 1 to 1000 by parallelizing in groups of 250:
  – 1 to 250
  – 251 to 500
  – 501 to 750
  – 751 to 1000

• Four threads

lastNumber = 1,000
sliceSize = 250

Pthreads

https://www.google.com/search?q=pthreads
https://computing.llnl.gov/tutorials/pthreads/
Prime Number w/Pthreads

SV Code
module top();
  import "DPI-C" function int eratosthenesBlockwise(longint lastNumber, longint sliceSize);
  found = eratosthenesBlockwise(lastNumber, sliceSize);
endmodule

C Code
int eratosthenesBlockwise(long lastNumber, long sliceSize) {
  pthread_t threads[100];
  from_to_t from_to[100];

  for (long from = 2; from <= lastNumber; from += sliceSize) {
    ...
    from_to[nthreads]= ...; // Initialize thread arguments
    rc = pthread_create(&threads[nthreads], NULL,
                        LaunchThread, &from_to[nthreads]);
    nthreads++;
  }
  for (int j = 0; j < nthreads; j++) {
    pthread_join(threads[j], &status); // Effectively a wait on thread 'j'
  }
}

DPI Call

10 threads

lastNumber = 1,000,000,000
sliceSize = 100,000,000

Break the range up into pieces

One piece per thread
Prime Number w/Pthreads (2)

http://create.stephan-brumme.com/
With adaptations from OpenMP to Pthreads

C Code
```c
typedef struct {
   long from, to, found;
} from_to_t;

void *LaunchThread(void *v) {
   from_to_t *from_to;
   from_to = (from_to_t *)v;
   from_to->found = eratosthenesOddSingleBlock(from_to->from, from_to->to);
}
```

Just a C Call

C Code
```c
// Run the algorithm between from and to.
int eratosthenesOddSingleBlock(const long from, const long to) {
 ...
}
```

<table>
<thead>
<tr>
<th># Threads</th>
<th>Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
</tr>
</tbody>
</table>

Original Code – out-of-the-box
Threads – to return or not

1. SV calls C code
2. C threads start
3. C threads end
4. C code returns to SV
5. No C threads still running

Figure 3: Pthreads started, compute, end

1. SV calls C code
2. C threads are continuously running
3. Communication between the running thread and the C call
4. C code returns to SV
5. C threads still running

Figure 4: Pthreads running - communicating via shared data
Other DPI usage
SystemC Golden Model

- SystemVerilog Testbench
- SV Interface
- RTL DUT
- SystemC Model

Instantiation and wire connection
SystemC
Golden Model - DPI

NOT ADVISED
Quite complicated
Hard to maintain
Non-standard

Some guy, Rich, wrote this code. He’s gone.

SystemVerilog Testbench

SV Interface

RTL DUT

SystemC Model

Special DPI API

Instantiation and wire connection

Threaded, special DPI calls
**SystemC Golden Model - TLM**

ADVANCED USAGE

- TLM can be complex
- Hard to maintain
- May be Non-standard

RTL DUT

SystemVerilog Testbench

SV Interface

Fancy TLM connector

SystemC Model

UVM Connect?

Fancy TLM connection

Instantiation and wire connection

3/1/2022
Rich Edelman, Mentor Graphics
SystemC – Golden Model – Direct Instantiation

ADVISED
Simple use-model
Easy to maintain
May be standard

Instantiation and wire connection
• DPI is cool. And fast.
• Data
  – Be aware - **Pass by value** versus **Pass by reference**
• Threads
  – **SV Threading** is “cooperative multi-threading”
    • It “seems” like it is threaded
  – **Pthreads** works well…
    • Understand your algorithm execution model
  – **SystemC integration** - Instantiate if possible
• All code available from the authors

Search the web for SV DPI Examples and Tips → **MANY Resources**