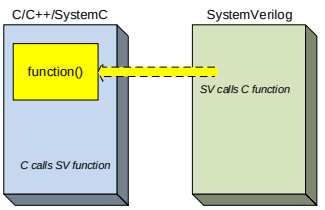
DPI: SV calls C function

2020

Ando Ki, Ph.D.
adki@future-ds.com

Table of contents

- SV calls C function with no return value
- SV calls C function with return value
- SV calls C function with array-in argument
- SV calls C function with array-out argument
- SV calls C function with struct-in argument
- SV calls C function with struct-out argument



Syntax import method

import {“DPI” | “DPI-C”} [context | pure] [*sv_local_func_name* =]
[function | task] [*c_func_name*]([*port_list*]);

- import
- DPI or DPI-C
- context: may cause side effect by the C routine
- pure: returns value and cause no side effect
- sv_local_func_name
- function
- task
- c_func_name

SV calls C function that has no return value

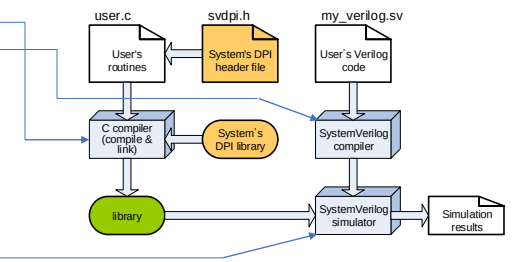
```
// c/function.c
#include "svdpi.h"
#include <stdio.h>

void myCFunction()
{
    printf("Hello from C function!\n");
}
```

```
// verilog/file.sv
module top;
    import "DPI-C" function void myCFunction();

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

```
$ xsc -o mydpi.so c/function.c
$ xelab -svlog verilog/file.sv -sv_lib mydpi
$ xsim top –runall
```



- xsc: Xilinx C compiler
- xelab:
- xsim:
- mydpi: user DPI library

SV calls C function that has no return value

- Go to the example directory
  - $ cd ...../codes/01_simple_sv2c/xsim
- Run ‘make’
  - (do not forget to setup environment for simulation)
  - $ make

```
$ cd ...../codes/01_simple_sv2c/xsim
$ make
```


SV calls C function that has return value (1/3)

```
// c/function1.c
#include "svdpi.h"
int myCFunc1( void ) { return 5; }
```

Mind void input

```
// c/function2.c
#include "svdpi.h"
int myCFunc2(int A, int *B) {
    *B = A/2;
    return A*2;
}
```

Mind pointer input

```
// c/function4.c
#include "svdpi.h"
#include <math.h>
double mySin( double C ) { return sin(C); }
double myCos( double C ) { return cos(C); }
double myTan( double C ) { return tan(C); }
```

Mind double and multiple functions

SV calls C function that has return value (2/3)

```
// verilog/file.sv
module top;
    import "DPI-C" function int  myCFunc1();
    import "DPI-C" function int  myCFunc2( input int A, output int B );
    import "DPI-C" function real mySin( input real C);
    import "DPI-C" function real myCos( input real C);
    import "DPI-C" function real myTan( input real C);
    integer iA, iB, iC;  real dD, dE;
    initial begin
        iA = myCFunc1();
        $display("%m %d", iA);
        iC = myCFunc2(iA, iB);
        $display("%m %d %d %d", iA, iB, iC);
        dD = 3.1415/2.0;
        $display("%m sin:%f cos:%f tan:%f", mySin(dD), myCos(dD), myTan(dD));
        $finish;
    end
endmodule
```

Mind direction
(in terms of the function)

Multiple routines in a single C file.

SV calls C function that has return value (3/3)

```
# xsim/Makefile
SHELL=/bin/sh

DIR_C       = ../c
DIR_VERILOG = ../verilog
LIB_DPI     = mydpi

all:
        xsc -compile ${DIR_C}/function1.c
        xsc -compile ${DIR_C}/function2.c
        xsc -compile ${DIR_C}/function4.c
        xsc -shared -o ${LIB_DPI}.so
        xelab -svlog ${DIR_VERILOG}/file.sv -sv_lib ${LIB_DPI}
        xsim top –runall
```

SV calls C function array-in argument (1/3)

```
// c/function.c
#include "svdpi.h"
#include <stdio.h>

int myFunc(const svOpenArrayHandle v) {
    int i;
    int low = svLow(v, 1);
    int high = svHigh(v, 1);
    for(i=low; i<=high; i++) {
        printf("[%d]=%d ", i, *((int*)svGetArrElemPtr1(v, i)));
    }
    printf("₩n");
    return i;
}
```

Mind svdpi.h function for open array

Mind how to get index

Mind how to get value of element

SV calls C function array-in argument (2/3)

```
module top();
    import "DPI-C" function int myFunc(input int v[]);
    int arr[4];    int dynArr[];    int ret;
    initial begin
        arr = {4, 5, 6, 7};
        ret = myFunc(arr);
        if (ret==4) $display("%m OK ", ret);
        else        $display("%m Failed ", ret);

        dynArr = new[6];
        dynArr = {8, 9, 10, 11, 12, 13};
        ret = myFunc(dynArr);
        if (ret==6) $display("%m OK ", ret);
        else        $display("%m Failed ", ret);

        arr = {127, 255, 255, -128};
        ret = myFunc(arr);
        if (ret==4) $display("%m OK ", ret);
        else        $display("%m Failed ", ret);
    end
endmodule
```

Mind how to pass array

Mind how to make dynamic array

SV calls C function array-in argument (3/3)

```
SHELL=/bin/sh

DIR_C       = ../c
DIR_VERILOG = ../verilog
LIB_DPI     = mydpi

all:
        xsc -compile ${DIR_C}/function.c
        xsc -shared -o ${LIB_DPI}.so
        xelab -svlog ${DIR_VERILOG}/file.sv -sv_lib ${LIB_DPI}
        xsim top -runall
```


SV calls C function array-out argument (1/3)

```
#include "svdpi.h"

int myFunc(svOpenArrayHandle v)
{
    int i;
    int low = svLow(v, 1);
    int high = svHigh(v, 1);
    for(i=low; i<=high; i++) {
        *((int*)svGetArrElemPtr1(v, i)) = 100+i;
    }
    return i;
}
```

Mind svdpi.h function for open array

Mind how to get index

Mind how to get pointer of element

SV calls C function array-out argument (2/3)

```
module top();
    import "DPI-C" function int myFunc(output int v[]);

    int dynArr[];
    int ret, idx;
    initial begin
        dynArr = new[6];
        ret = myFunc(dynArr);
        if (ret!=6) $display("%m Failed ", ret);
        else begin
            for (idx=0; idx<ret; idx=idx+1)
                $display("%2m [%4d]=%d", idx, dynArr[idx]);
        end
    end
endmodule
```

Mind how to pass array

Mind how to make dynamic array

SV calls C function array-out argument (3/3)

```
SHELL=/bin/sh

DIR_C       = ../c
DIR_VERILOG = ../verilog
LIB_DPI     = mydpi

all:
        xsc -compile ${DIR_C}/function.c
        xsc -shared -o ${LIB_DPI}.so
        xelab -svlog ${DIR_VERILOG}/file.sv -sv_lib ${LIB_DPI}
        xsim top -runall
```

SV calls C function struct-in argument (1/3)

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

typedef struct pkt_t {
    char   A;
    int    B;
    float  C;
    double D;
} pkt_t;

int myFunc(pkt_t *v)
{
    printf("%s() A=%c B=%d C=%f D=%f₩n", __func__, v->A, v->B, v->C, v->D);
    return 0;
}
```


SV calls C function struct-in argument (2/3)

```
typedef struct {
    byte        A;
    int         B;
    shortreal   C;
    real        D;
} pkt_t;

module top();
    import "DPI-C" function int myFunc(inout pkt_t v);
    pkt_t  pkt;
    int ret;
    initial begin
       pkt.A = 8'h41;
       pkt.B = 100;
       pkt.C = 100.1;
       pkt.D = 1.3e10;
       ret = myFunc(pkt);
       #1; $write("₩n");
       if (ret==0) $display("%m OK ", ret);
       else        $display("%m Failed ", ret);
    end
endmodule
```

SV calls C function struct-in argument (3/3)

```
SHELL=/bin/sh

DIR_C       = ../c
DIR_VERILOG = ../verilog
LIB_DPI     = mydpi

all:
        xsc -compile ${DIR_C}/function.c
        xsc -shared -o ${LIB_DPI}.so
        xelab -svlog ${DIR_VERILOG}/file.sv -sv_lib ${LIB_DPI}
        xsim top -runall
```


SV calls C function struct-out argument (1/3)

```
#include "svdpi.h"

typedef struct pkt_t {
    char   A;
    int    B;
    float  C;
    double D;
} pkt_t;

int myFunc(pkt_t *v)
{
    v->A = 'P';
    v->B = 111;
    v->C = 123.321;
    v->D = 1.2;
    return 0;
}
```

SV calls C function struct-out argument (2/3)

```
module top();
    typedef struct {
        byte        A;
        int         B;
        shortreal   C;
        real        D;
    } pkt_t;

    import "DPI-C" function int myFunc(inout pkt_t v);

    pkt_t  pkt;      int ret;
    initial begin
       ret = myFunc(pkt);
       #1; $write("₩n");
       if (ret!=0) $display("%m Failed ", ret);
       else begin
            $display("%m pkt.A = %c", pkt.A);
            $display("%m pkt.B = %d", pkt.B);
            $display("%m pkt.C = %f", pkt.C);
            $display("%m pkt.D = %f", pkt.D);
       end
    end
endmodule
```


SV calls C function struct-out argument (3/3)

```
SHELL=/bin/sh

DIR_C       = ../c
DIR_VERILOG = ../verilog
LIB_DPI     = mydpi

all:
        xsc -compile ${DIR_C}/function.c
        xsc -shared -o ${LIB_DPI}.so
        xelab -svlog ${DIR_VERILOG}/file.sv -sv_lib ${LIB_DPI}
        xsim top -runall
```