

# Detailed DPI Example Code Fragments

Created by John Stickley

Edited by Doug Warmke

May 17, 2003

This document contains detailed examples of using DPI import and export functions. The intention is to demonstrate a number of features of DPI, including:

- Mapping of types between C and SystemVerilog
- Trade-offs when choosing which types to use
- Use of export functions
- Use of context import functions and user data

There are two variations of similar functionality presented. The functionality involves passing an ethernet packet header across the SV/C language boundary, and doing various processing on the packet in both languages.

Variation 1:

Direct use of native C struct for input arguments and function return – binary compatible

Variation 2:

Direct use of integers and large and small packed array bit vector inputs and outputs - source compatible

## Variation 1 – Struct composed of Native C types

In the following example, an ethernet packet header struct is directly passed to an exported SV function. A computed CRC is returned as a function return argument and compared.

Here's the C definition of the header:

```
struct etherHeaderT {
    unsigned type;
    unsigned length;
    unsigned long long dest_addr;
    unsigned long long src_addr;
};
```

The same struct and exported HDL function is declared on the SV side as follows:

```
typedef struct {
    int unsigned type;           // 16 bit type
    int unsigned length;        // 16 bit length
    longint unsigned dest_addr; // 48 bit dest_addr
    longint unsigned src_addr;  // 48 bit src_addr
} etherHeaderT;

export "DPI" SendPacketHeader=handlePacketHeader;

// Returns 32 bit CRC
function int unsigned handlePacketHeader(
    input etherHeaderT header);
    ...
    return computedCRC;
endfunction
```

The SV function will return a computed CRC as a return arg.

The required prototype for the C function wrapper is as follows:

```
unsigned SendPacketHeader(           // function int unsigned
    const etherHeaderT* header);    // input etherHeaderT header
```

Here is a scenario for how this function might be called from a C model. This code is binary compatible across all implementations:

```
void EthPort::SendHeader(const etherHeaderT* header, unsigned crc) {
    if (crc != SendPacketHeader(header))
        logError("CRC mismatch");
}
```

## Variation 2 – Use of large and small packed array types

In the following example, a "DPI" import function is used to alert a C model that an ethernet packet with a certain src, dest, and crc is available in a slave port's output queue. The C model later requests the packet payload chunk by chunk, using a "DPI" export function. After all chunks are uploaded it compares the CRC.

In this case, all arguments are bit vectors. The CRC and length args are small bit vectors while the other arguments (src address, dest address, and packet payload) are large bit vectors.

The imported and exported HDL functions are declared on the SV side as follows:

```
import "DPI" context function void NotifyPacketAvailable(
    input bit [47:0] destAddr, // 48 bit dest_addr
    input bit [47:0] srcAddr,  // 48 bit src_addr
    input bit [15:0] length,   // 16 bit length
    input bit [31:0] crc );    // 32 bit crc

export "DPI" GetPayloadChunk = sendChunk;
function void sendChunk(
    input bit [47:0] srcAddr,
    output bit [511:0] payloadChunk,
    output unsigned int chunkLength);

    verifySrcAddress(srcAddr);
    payloadChunk = nextPayloadChunk;
    chunkLength = 512 < remainingLength ? 512 : remainingLength;
    remainingLength = remainingLength - chunkLength;
endfunction
```

The required prototype for the user supplied imported C function and the exported C function wrapper is defined according to the DPI is as follows:

```
extern "C" {

    // Imported DPI function
    void NotifyPacketAvailable(
        const svBitPackedArrRef destAddr, // input bit [47:0] destAddr;
        const svBitPackedArrRef srcAddr,  // input bit [47:0] srcAddr;
        const svBitVec32 length,          // input bit [15:0] length;
        const svBitVec32 crc );           // input bit [31:0] crc;

    // Exported DPI function
    void GetPayloadChunk(
        const svBitPackedArrRef srcAddr,  // input bit [47:0] srcAddr;
        svBitPackedArrRef payloadChunk,   // output bit [511:0] payloadChunk;
        unsigned* chunkLength );          // output unsigned int chunkLength;
};
```

A C++ model that uses these functions might be defined as follows:

```
#include "svdpi.h"
#include "svdpi_src.h"

class EthPort {
private:
    svScope dSvScope;
    bool dPacketAvailable;
    unsigned long long dDestAddr, dSrcAddr;
    unsigned dCrc;
    unsigned dLength;
    char dPayload[1500];

    void logError(const char* message);
    unsigned updateCrc(unsigned crc, char data);

public:
    EthPort(const char* pathname);
    void CheckForPacket();

friend void NotifyPacketAvailable(
    const svBitPackedArrRef destAddr, // input bit [47:0] destAddr;
    const svBitPackedArrRef srcAddr,  // input bit [47:0] srcAddr;
    const svBitVec32 length,          // input bit [15:0] length;
    const svBitVec32 crc);           // input bit [31:0] crc;
};
```

Notice how the imported function is declared as a friend of the C++ model class. This allows it to have access to the local C++ model context using the svPut/GetUserdata() access functions as will be shown below.

```
EthPort::EthPort(const char *pathname)
    : dPacketAvailable(false), dDestAddr(0LL), dSrcAddr(0LL),
      dCrc(0), dLength(0)
{
    dSvScope = svGetScopeFromName(pathname);
    svPutUserData(dSvScope, (void *)(&NotifyPacketAvailable), this);
}
```

Using the DPI function svGetScopeFromName(), the model constructor establishes a scope context for the SystemVerilog module instance that acts as the C++ model's counterpart on the HDL side. This only needs to happen at construction time.

In addition, the 'this' pointer of the C++ model is associated with the HDL module scope. This way, when imported functions are called from the HDL module, the local C++ model context can be obtained using the svGetScope() and svGetUserData() functions. This is illustrated in the implementation of the imported C function:

```
void NotifyPacketAvailable(
    const svBitPackedArrRef destAddr,
    const svBitPackedArrRef srcAddr,
    const svBitVec32 length,
    const svBitVec32 crc)
{
    EthPort *me = (EthPort*)svGetUserData(
        svGetScope(), &NotifyPacketAvailable);

    me->dPacketAvailable = true;
    me->dDestAddr = svGet64Bits(destAddr, 0);
    me->dSrcAddr = svGet64Bits(srcAddr, 0);
    me->dCrc = crc;
    me->dLength = length;
}
```

At some later point in time, the C testbench might invoke a method in the C model to check to see if a packet notification had occurred. If so, the code would enter a loop that calls the exported DPI function to retrieve payload chunks until the entire payload is received. The computed CRC is then checked at the end:

```
void EthPort::CheckForPacket()
{
    if (dPacketAvailable == true) {
        dPacketAvailable = false;
        svSetScope(dSvScope);

        unsigned crc, i, chunkLength;
        SV_BIT_PACKED_ARRAY(48, srcAddr);
        SV_BIT_PACKED_ARRAY(512, payloadChunk);

        svPutPartSelectBit(srcAddr, (svBitVec32)(dSrcAddr), 0, 32);
        svPutPartSelectBit(srcAddr, (svBitVec32)(dSrcAddr>>32), 32, 16);

        // Get payload chunks from SV side for queued packet matching
        // src address until entire payload is retrieved.
        for (i = 0; i < dLength; i) {
            GetPayloadChunk(srcAddr, payloadChunk, &chunkLength);

            for (unsigned j = 0; j < chunkLength / 8; i++, j++) {
                dPayload[i] = svGetBits(payloadChunk, j * 8, 8);
                crc = updateCrc(crc, dPayload[i]);
            }
        }
        if (crc != dCrc)
            logError("CRC mismatch");
    }
}
```

} }