SV to C, C to SV Function Call Proposal

Joao Geada, John Stickley

Contents

- SV-to-C Function and Task Calls
  - Declaration Syntax
  - Free Function Binding
  - Context Specific Function Binding
- C-to-SV Function and Task Calls
  - Declaration Syntax
  - Context Specific Task Binding
- Packet Router Example
  - SystemC Testbench Root
  - SystemC EthPortWrapper Module
  - SystemVerilog EthPort Module

SV-to-C Declaration Syntax

```
extern_decl ::=
    extern access_mode ? attribute (, attribute) * ?
      function_decl | task fname ( extern_func_args ? );
function_decl ::= function return-type

access_mode ::= ( "A" | "C" )

attribute ::= pure | context
```

Examples:

```
// Declare an SV callable, pure,
// free standing(non-context specific), C function
extern "C" pure function int MyCFunc( input int portID );

// Declare an SV callable, context specific, C task
extern "C" context task MyCTask(
   input int portID, output int mappedID );
```

SV-to-C Free Function Binding

- For free functions, i.e. no **`context`** keyword in declaration, no special binding API call required.
- Simply define the function on the C side …

```
int MyCFunc( int portID ){
    return map(portID);
}
```

- … and call it from the SV side.

```
always @( my_clock ) begin
       if( my_reset ) begin
           // Blah, blah, blah;
       end
       else begin
           if( state == READY ) begin
               mappedID <= MyCFunc( portID ); // Call to C.
               state <= WAITING;
           end
       // Blah, blah, blah;
    end
```

SV-to-C Context Specific Function Binding

- For context specific functions or tasks use `tf_isetworkarea()`, `tf_igetworkarea()`, `tf_getinstance()`, `tf_mipname()` in current PLI interface to establish context to be associated with each instance specific SV call.
- The equivalent can probably also be done using VPI functions as well (exercise left to reader !).
- Define a function on the C side that calls `tf_getinstance()` and `tf_igetworkarea()` to fetch a *context* pointer previously established as described above:

```
void MyCTask(int portID, int *mappedID ){
    MyCModel *me = (MyCModel *)tf_igetworkarea(
        tf_getinstance() );
    *mappedID = me->map(portID);
}
```

- During initialization, bind the model context to the SV caller instance:

```
void MyCModel::Init( ){
    tf_isetworkarea( this, tf_mipname("top.u1") );
}
```

C-to-SV Declaration Syntax

```
export_decl ::=
    export access_mode ? attribute (, attribute) * ?
      function | task fname;

access_mode ::= ( "A" | "C" )

attribute ::= pure | context
```

Example:

```
// Declare a C callable, context specific, SV task
export "C" context task MySvTask;

task MySvTask;
    input int portID;
    output int mappedID;

    mappedID = map(portID);
endtask
```

C-to-SV Context Specific Task Binding

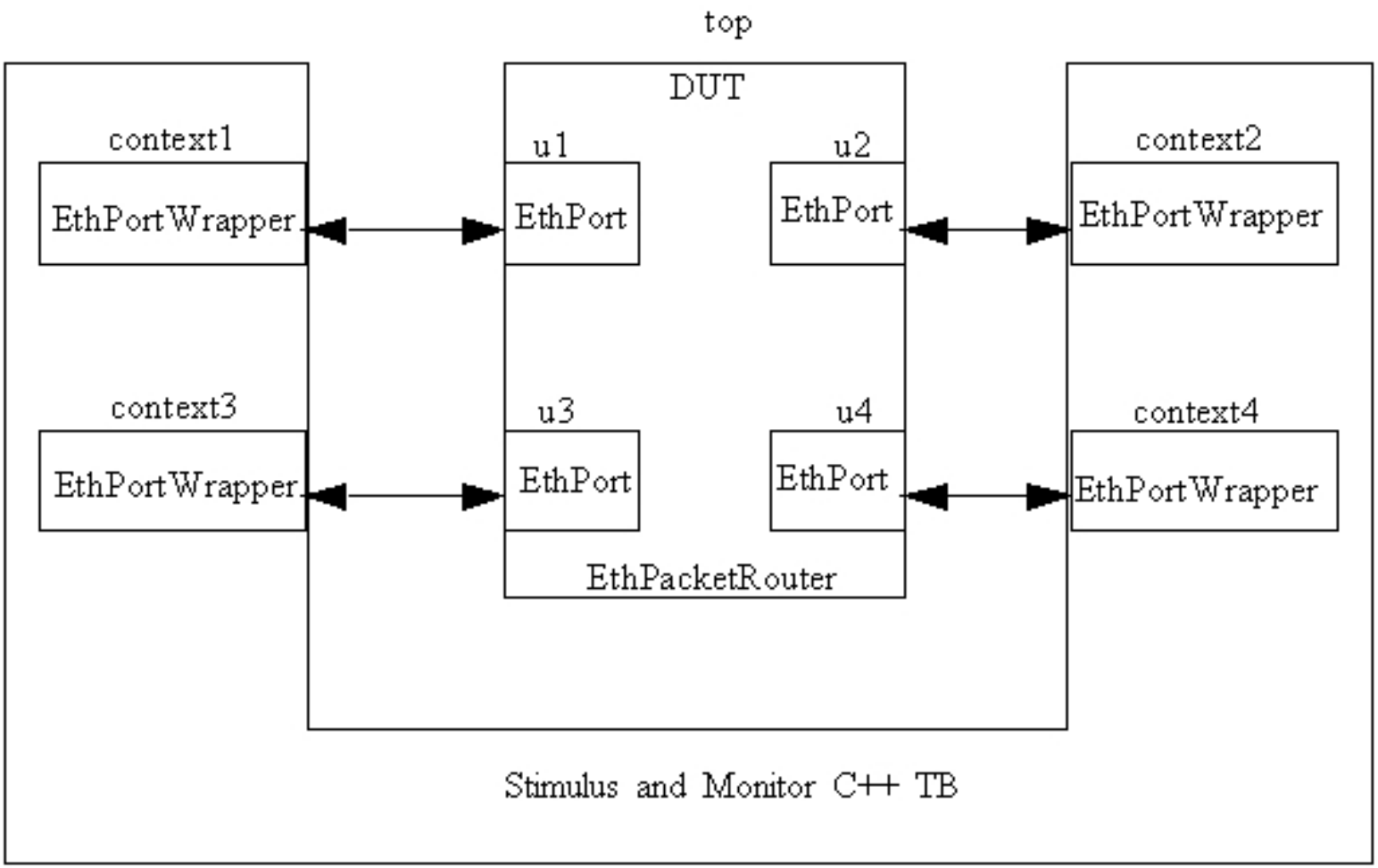
- For context specific functions or tasks use `tf_mipname()` to establish context to be associated with each instance specific SV callee:
- The equivalent can probably also be done using VPI functions as well (exercise left to reader !).
- During initialization, establish a context handle to the SV callee instance:

```
void MyCModel::Init( ){
    svContext = tf_mipname( “top.u1.MySvTask” );
}
```

- Now at run-time call the SV function from C passing the context handle as the first argument:

```
void MyCModel::RunTest( int portID ){
    MySvTask( svContext, portID, &mappedID );
    . . .
}
```

Packet Router Example



Packet Router Example: SystemC Testbench Root

```
 1  SC_MODULE( TestBench ){
 2      private:
 3          EthPortWrapper *context1;
 4          EthPortWrapper *context2;
 5          EthPortWrapper *context3;
 6          EthPortWrapper *context4;
 7          int numOutputs;
 8
 9          void testThread(); // Main test driver thread.
10      public:
11          SC_CTOR( System ) : numOutputs(0) {
12
13              SC_THREAD( testThread );
14              sensitive << UTick;
15
16              // Construct 4 instances of reusable EthPortWrapper
17              // class for each of 4 different HDL module instances.
18              context1 = new EthPortWrapper( "c1" ); context1->Bind( "top.u1", this );
19              context2 = new EthPortWrapper( "c2" ); context2->Bind( "top.u2", this );
20              context3 = new EthPortWrapper( "c3" ); context3->Bind( "top.u3", this );
21              context4 = new EthPortWrapper( "c4" ); context4->Bind( "top.u4", this );
22          }
23          void BumpNumOutputs(){ numOutputs++; }
24  };
25
26  void TestBench::testThread(){
27      // Now run a test that sends random packets to each input port.
28      context1->PutPacket( generateRandomPayload() );
29      context2->PutPacket( generateRandomPayload() );
30      context3->PutPacket( generateRandomPayload() );
31      context4->PutPacket( generateRandomPayload() );
32
33      while( numOutputs < 4 ) // Wait until all 4 packets have been received.
34          sc_wait();
35  }
```

Packet Router Example: SystemC EthPortWrapper Module

```
 1  SC_MODULE( EthPortWrapper ){
 2      private:
 3          void *svContext;
 4          sc_module *myParent;
 5
 6      public:
 7          SC_CTOR( EthPortWrapper ) : svContext(0), myParent(0) { }
 8          void Bind( const char *hdlPath, sc_module *parent );
 9          void PutPacket( vec32 *packet );
10
11      friend void HandleOutputPacket( int portID, vec32 *payload );
12  };
13
14  void EthPortWrapper::Bind( const char *hdlPath, sc_module *parent ){
15      myParent = parent;
16      svContext = tf_mipname( hdlPath );
17      tf_isetworkarea( this, tf_mipname(hdlPath) );
18  }
19
20  void EthPortWrapper::PutPacket( vec32 *packet ){
21      PutPacket( svContext, packet ); // Call SV function.
22  }
23
24  friend void HandleOutputPacket( int portID, vec32 *payload ){
25      EthPortWrapper *me = (EthPortWrapper *)tf_igetworkarea( tf_getinstance() );
26
27      me->myParent->BumpNumOutputs(); // Let top level know another packet received.
28
29      printf( "Received output on port on port %\n", portID );
30      me->DumpPayload( payload );
31  }
```

Packet Router Example: SystemVerilog EthPort Module

```
 1  module EthPort(
 2          MiiOutData,                     MiiInData,
 3          MiiOutEnable,                   MiiInEnable,
 4          MiiOutError,                    MiiInError,
 5          clk, reset );
 6
 7          input [7:0] MiiOutData;         output [7:0] mii_data;
 8                                          reg    [7:0] mii_data;
 9          input MiiOutEnable;             output mii_enable;
10                                          reg    mii_enable;
11          input MiiOutError;              output mii_error;
12                                          reg    mii_error;
13          input clk, reset;
14
15      extern "C" context task HandleOutputPacket(
16          input int portID, input reg [1439:0] *payload );
17
18      export "C" context task PutPacket;
19
20      reg packetReceivedFlag;
21      reg [1499:0] packetData;
22
23      task PutPacket;
24          input reg [1499:0] packet;
25
26          packetData = packet;
27          packetReceivedFlag = 1;
28      endtask
```

Packet Router Example: SystemVerilog EthPort Module (cont'd)

```
 1      always @( clk ) begin      // input packet FSM
 2          if( reset ) begin
 3              // Blah, blah, blah ...
 4          end
 5          else begin
 6              if( instate == READY ) begin
 7                  if( packetReceived )
 8                      state <= PROCESS_INPUT_PACKET;
 9              end
10              else if( instate == PROCESS_INPUT_PACKET ) begin
11                  // Start processing input packet byte by byte ...
12              end
13          end
14      end
15
16      always @( clk ) begin     // output packet FSM
17          if( reset ) begin
18              // Blah, blah, blah ...
19          end
20          else begin
21              if( outstate == READY ) begin
22                  if( MiiOutEnable )
23                      state <= PROCESS_OUTPUT_PACKET;
24              end
25              else if( outstate == PROCESS_OUTPUT_PACKET ) begin
26                  // Start assembling output packet byte by byte ...
27                  ...
28                  // Make call to C side to handle it.
29                  HandleOutputPacket( myPortID, outPacketVector );
30              end
31          end
32      end
33  endmodule
```