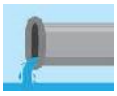
SCE-MI

standard co-emulation modeling interface

Bridge untimed HVL and timed HDL

- Huge speed differential: HDL on emulator and HVL on server
  - Extensive use of import/export functions becomes bottleneck
    - Blocking calls
  - Divide test bench into HVL and HDL (BFM) components
    - send one transaction that will control 100s of clock cycles
  - Limited use of bandwidth between server and emulator
- Streaming transactions through **pipes** vastly improves perf
  - Extension to SystemVerilog DPI standard
  - Implementations may include vendor specific optimization

SCE-MI Transaction Pipes



Unidirectional channel between HVL and HDL

- Input pipe from HVL to HDL, output pipe from HDL to HVL
- Similar to one-way transaction channels like SystemC TLM
- Unix stream model: each pipe is uniquely named using hierarchical path

- Wide: Payload up to 1k fixed sized bit vector elements

- Deep: Buffered to increase throughput
  - Buffer depth defined by implementation
  - Transaction ordering guaranteed
  - Can flush (useful for synchronization between HVL and HDL)

- Operations via simple function calls
  - Blocking: *send, receive, flush*
    - return after operation completes
  - Non-blocking: *try_send*, *try_receive*
    - may return without operation being performed

Transaction Pipe Definition

- HDL: *scemi_input_pipe* or *scemi_output_pipe* instance

  Element Size: multiple of bytes

  Payload Max Elements: max HDL side can send/receive

  Buffer Depth: max elements pipe can hold
  - Automatic data transfer occurs when close to full

```
HDL side

scemi_input_pipe
 #(.BYTES_PER_ELEMENT(<>),
   .PAYLOAD_MAX_ELEMENTS(<>),
   .BUFFER_MAX_ELEMENTS(<>))
    <pipe_name> ();

scemi_output_pipe
 #(.BYTES_PER_ELEMENT(<>),
   .PAYLOAD_MAX_ELEMENTS(<>),
   .BUFFER_MAX_ELEMENTS(<>))
    <pipe_name> ();
```

- HVL
  - Handle derived from HDL name

```
C side
#include "scemi_pipe.hxx"

void *scemi_pipe_c_handle(
    const char* <name_path>);
```

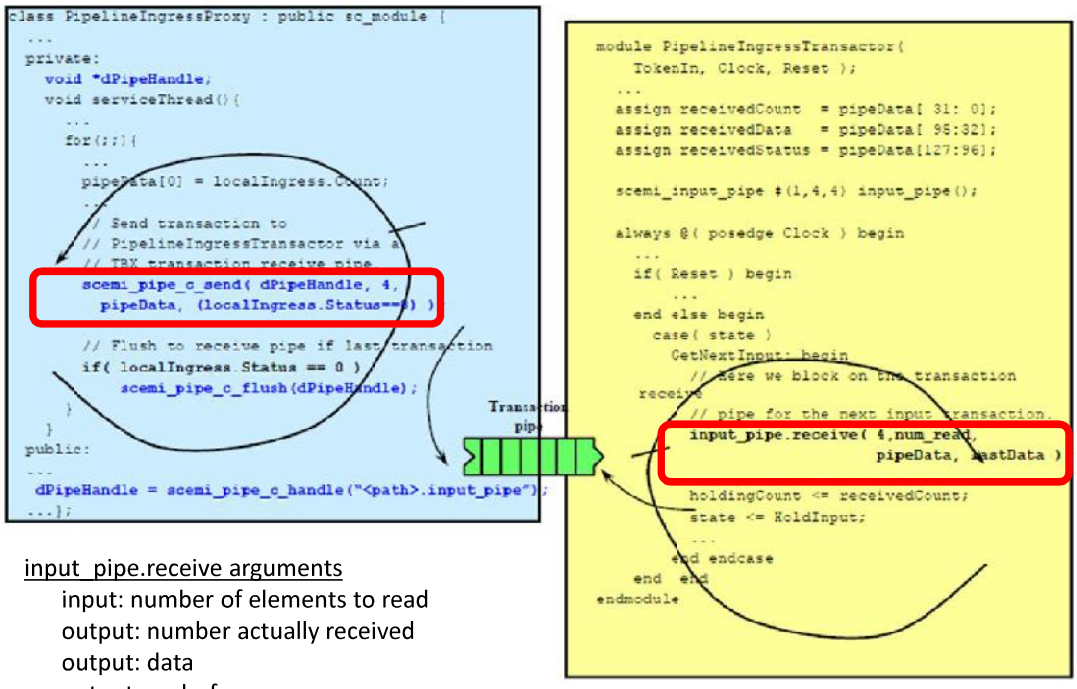

Transaction *Input* Pipes

```
HDL-side
<pipe_name>.receive(
   input  int num_elements,
   output int num_elements_valid,
   output bit [MAX_DATA-1:0] data,
   output bit eom );
```

```
C-side
void scemi_pipe_c_send(
    void *pipe_handle,
   int num_elements,
    const svBitVecVal *data,
    svBit eom );
```

- HDL side, transactions are **received** via interface task receive()
- Interface task aregument list:
  - num_elements - # elements to be read
  - num_elements_valid - # valid elements
  - data - Data payload
  - eom – End-of-Message marker flag
- C side, transactions are **sent** via function calls with the fixed name, scemi_pipe_c_send()

Transaction Input Pipe



input_pipe.receive arguments

- input: number of elements to read
- output: number actually received
- output: data
- output: end-of-message

Transaction *Output* Pipes

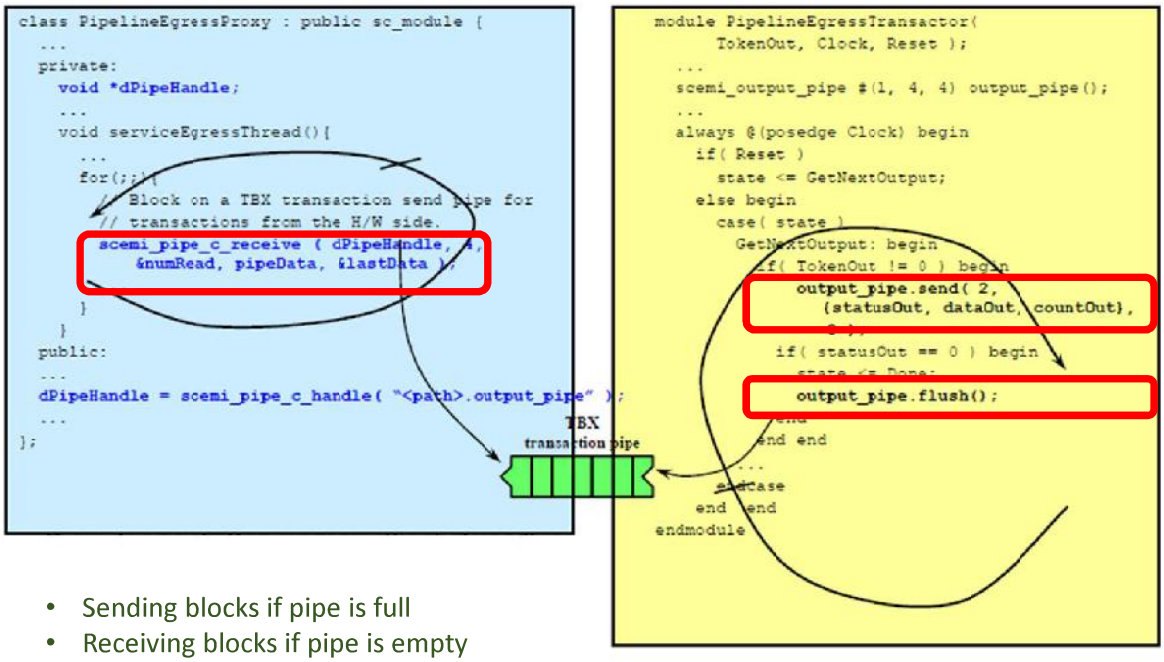
```
HDL-side
<pipe_name>.send(
   input int num_elements,
   input bit [MAX_DATA-1:0] data,
   input bit eom );
```

- HDL side, transactions are sent via interface taks send()
- Interface argument list:
  - num_elements - # elements to be written
  - data - Data payload
  - eom – End-of-Message marker flag

```
C-side
void scemi_pipe_c_receive(
   void *pipe_handle,
   int num_elements_requested,
   int *num_elements_valid,
   svBitVecVal *data,
   svBit *eom );
```

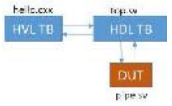
- C side, transactions are received via function calls with the fixed name, scemi_pipe_c_receive()

Transaction Output Pipe



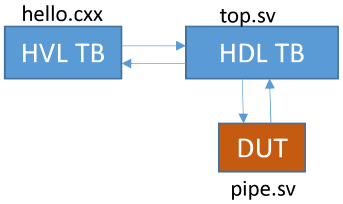
- Sending blocks if pipe is full
- Receiving blocks if pipe is empty

Hello_world example



- HVL: read content of file and send to HDL
- HDL:
  - transactor: provide data from HVL to DUT
    provide results from DUT to HVL
  - DUT: process data from transactor
    return the result to HDL-transactor
- Data transfer between HDL-transactor and HVL streaming
  - transaction pipes best suited to task

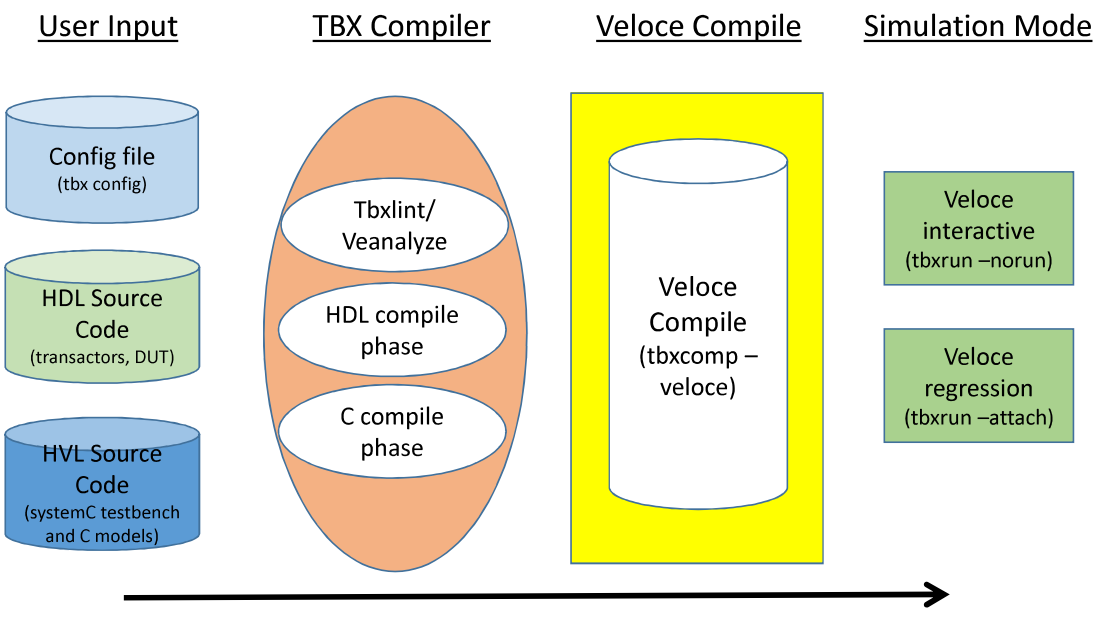
Mentor Graphics hello world example



- **HDL testbench: hdl/top.sv**
  - generates clock/reset
  - fetches byte stream from HVL and pushes into DUT (pipe)
  - monitors DUT output,
    - collects it in a buffer
    - sends the outcoming bytes (maximum 40 at a time) to HVL
  - terminates using $finish when all bytes are received
- **DUT: hdl/pipe.sv**
  - Is essentially a pipeline of registers (one byte wide)
  - The pipeline depth is parameterized. (depth = 1024 here)
- **HVL testbench: hvl/hello.cxx**
  - HDL centric test-environment
  - Provides definitions for the two DPI import calls
    - DPI import 'getbuf'
      - reads a file 'msg' and sends the character bytes in (maximum 40 at a time)
    - DPI import 'sendbuf'
      - receives the bytes back and prints them on screen.

```
$VMW_HOME/examples
hello.cxx
top.sv
pipe.sv
```


TBX – Veloce Compilation Flow



Makefile.TBX

```
all: compile ccomp sim check clean

compile:
    tbxlib work
    tbxmap work ./work
    veanalyze hdl/pipe.sv hdl/top.sv
    tbxcomp -top top

ccomp
    tbxcomp -c -cfiles hvl/hello.cxx

sim:
    tbxrun | tee transcript.tbx
    grep HVL transcript.tbx > result

check:
    diff -w result result.gold

clean:
    rm -rf work tbx.dir tbxsim.v tbx.map
tbxbindings.h* debussy.cfg dpibindings.h
transcript.tbx *.log Recompiled.list tree.out
vsim.wlf transcript result TRACE.txt dmslogdir
```

Many preset environment variables (use printenv to view)

tbx.config

```
rtlc -partition_module_xrtl top
velsyn -D1S
```

See appendix

Makefile.MTI (for ModelSim)

```
all: compile ccomp sim check clean

compile:
    vlib work
    vmap work ./work
    vlog hdl/pipe.sv hdl/top.sv -dpiheader tbxbindings.h

ccomp:
    g++ -shared -o dpi.so -g hvl/hello.cxx -I. -I$(MGC_HOME)/include

sim:
    MTI_VCO_MODE=32; export MTI_VCO_MODE; \
    vsim -c top -sv_lib dpi -do "run -all; quit -f"
    grep -w HVL transcript | sed 's/^#//g' > result

check:
    diff -w result result.gold

clean:
    rm -rf work tbxbindings.h transcript vsim.wlf dpi.so result
```


tbx.config Sample file for Veloce

- **XRTL options**

  rtlc –xrtl
  rtlc -partition_rtl Top
  rtlc -partition_module_xrtl Top
  rtlc -partition_module_xrtl Producer

- **RTLC options**

  rtlc -max_error_count 10
  rtlc -allow_4ST
  rtlc -allow_FRN -allow_4ST
  rtlc -no_compile_celldefines

- **Specifying Simulator**

  tbxsim -simtype mti

- **Velsyn and Velgs options**

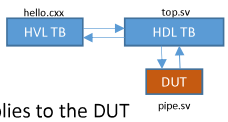
  velsyn GLOB_PLATFORM=D1S
  velsyn -Dump -e50 long_paths.dump
  velsyn -SrND -Mm 8.7
  velgs -crit

Mentor Graphics hello world example

$TBX_HOME/examples
hello.cxx
top.sv
pipe.sv

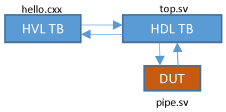
```
/*
 *  Mentor Graphics Corporation
 *
 *  Copyright 2003-2007 Mentor Graphics Corporation
 *  All Rights Reserved
 *
 *  THIS WORK CONTAINS TRADE SECRET AND PROPRIETARY
 *  INFORMATION WHICH IS THE PROPERTY OF MENTOR
 *  GRAPHICS CORPORATION OR ITS LICENSORS AND IS
 *  SUBJECT TO LICENSE TERMS.
 */

//-------------------------------------------------------------------------
//  Mentor Graphics, Corp.
//
//  (C) Copyright, Mentor Graphics, Corp. 2003-2004
//  All Rights Reserved
//  Licensed Materials - Property of Mentor Graphics, Corp.
//
//  No part of this file may be reproduced, stored in a retrieval system,
//  or transmitted in any form or by any means --- electronic, mechanical,
//  photocopying, recording, or otherwise --- without prior written permission
//  of Mentor Graphics, Corp.
//
//  WARRANTY:
//  Use all material in this file at your own risk.  Mentor Graphics, Corp.
//  makes no claims about any material contained in this file.
//-------------------------------------------------------------------------
```


Mentor Graphics hello world example



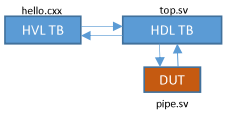

```
module top;

// Clock/Reset generation logic.

  reg clock, reset = 1;
  //tbx_clkgen
  initial begin
    clock = 0;
    forever #5
      clock = ~clock;
  end

  //tbx_clkgen
  initial begin
    reset = 1;
    #100 reset = 0;
  end

// DUT instantiation
// pipeline of registers that is 1024 deep and 8 bit wide.

  reg [7:0] inbyte = 8'hff;
  wire [7:0] outbyte;
  pipe #(1024) dut (outbyte, inbyte, clock, reset);

// DPI Import 'getbuf' :  get the byte stream from HVL.

  import "DPI-C" function void getbuf(output bit [319:0] stream,
                                      output int count,
                                      output bit eom);

// XRTL FSM
// fetches the byte stream from HVL applies to the DUT
// one each clock. It stops once it receives eom.

  bit eom = 0;
  int remaining = 0;
  bit [319:0] stream;

  always @(posedge clock) begin
    if(reset)
      inbyte <= 8'hff;

    else begin
        if(remaining==0 && eom==0)
            getbuf(stream, remaining, eom);
        if(remaining > 0) begin
            inbyte <= stream[7:0];
            stream = stream >> 8;
            remaining--;
        end

    end
  end

// DPI import 'sendbuf' : send DUT output stream of bytes to HVL

  import "DPI-C" function void sendbuf (
                  input bit [319:0] buffer,
                  input int count);
```

Mentor Graphics hello world example




```
// XRTL FSM
//  captures the output character stream and sends it to the HVL testbench in large packets.
//       - packet size = 40 bytes
//       - bytes with the value 8'hff are ignored
//       - byte with value '0' is taken as the end of stream
//       - makes use of a utility function 'sendbyte' which packs the bytes into a buffer of 40 bytes and sends it to HVL

   int loc = 0;
   bit [319:0] buffer = ~(320'h0);

   function void sendbyte (input bit [7:0] b);
   begin
      buffer = buffer << 8;
      buffer [7:0] = b;
      loc=loc+1;
      if(loc==40 || b==0) begin
         sendbuf(buffer, loc);
         buffer = ~(320'h0);
         loc = 0;
      end
   end
   endfunction

   always @(posedge clock)
      if(!reset) begin
         if(outbyte!=8'hff)
            sendbyte(outbyte);
         if(outbyte==0)
            $finish;
      end

endmodule
```


Mentor Graphics hello world example

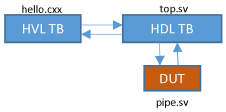



```
module pipe (outbyte, inbyte, clock, reset);
parameter DEPTH = 8;
output [7:0] outbyte;
input [7:0] inbyte;
input clock, reset;

// Generate segments:
//      - Each segment adds one level of registers
//      - Add as many segments as DEPTH
//      - The pipeline consists of
//              a first segment,
//              a series of mid segments and then
//              a last segment.

genvar i;

generate begin : segments
  wire [7:0] tmp [0:DEPTH-2];

  for(i=0;i<DEPTH;i++) begin
      if (i==0)               segment first  (tmp[i],   inbyte,     clock, reset);
      else if(i<DEPTH-1)      segment mid    (tmp[i],   tmp[i-1],   clock, reset);
      else                    segment last   (outbyte, tmp[i-1],    clock, reset);
      end
   end
endgenerate

endmodule
```

```
module segment (outbyte, inbyte, clock, reset);
    output [7:0] outbyte;
     reg [7:0] outbyte;
     input [7:0] inbyte;
      input clock, reset;

always @(posedge clock) begin
   if(reset)
          outbyte = 8'hff;
   else
          outbyte = inbyte;
end

endmodule
```

Hello.cxx

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


//
// DPI import function 'sendbuf':
// - used to send the DUT output bytes back to HVL
//   which are then displayed on screen
//
void sendbuf  (const svBitVecVal* buffer, int count) {

  svBitVecVal b = 0;
  for(int i=count;i>0;i--) {
   svGetPartselBit(&b, buffer, (i-1)*8, 8);
   if(b==0) break;
   printf("%c", b);
   if(b=='\n') printf("HVL:");
  }
  if(b==0) printf("\nHVL: Complete message received.\n");
  fflush(stdout);
}
```




```
// DPI import function 'getbuf':
// used by HDL to fetch the input byte stream.
//  The first call opens the message file 'msg'.
// Subsequent calls read further into the file.
//  When the file ends, the argument 'eom' is  set to indicate end of data.
FILE* in_msg = NULL;

void getbuf  (svBitVecVal* buf, int* count, svBit* eom) {
  // Open file "msg" and start streaming in the bytes..
  if(!in_msg) {
     printf("HVL: Opening file \"msg\"..\n");
     in_msg = fopen("msg", "r");
  }
 char b;
  int i = 0;
  while((b = fgetc(in_msg)) != EOF) {
    svPutPartselBit(buf, b, 8*i, 8);
    if(i==39) {
       *count = 40;
       *eom = 0;
       printf("HVL: Sending 40 bytes..\n");
       return;
    }
    i++;
  }
// Send the remaining bytes with eom.
  svPutPartselBit(buf, 0, 8*i, 8);
  printf("HVL: Sending last %d bytes.\n", i+1);
  *count = i + 1;
  *eom = 1;

  return;
}
```

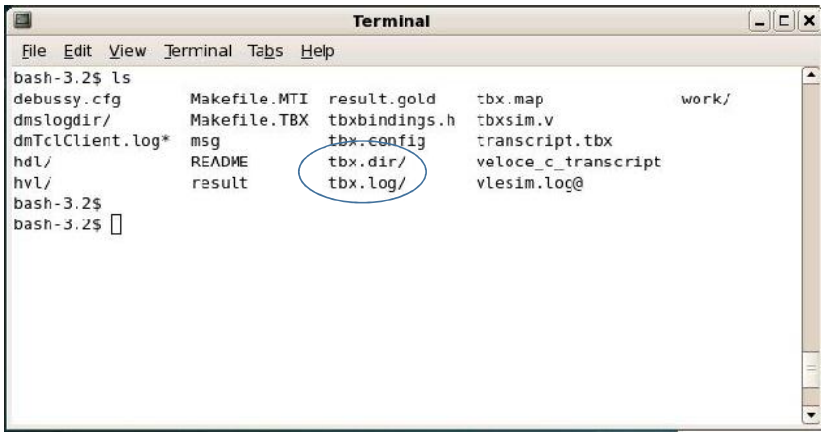


```
HVL: Opening file "msg"..
HVL: Sending 40 bytes..
HVL: Sending 40 bytes..
HVL: Sending 40 bytes..
HVL: Sending 40 bytes..
HVL: Sending 40 bytes..
HVL: Sending 40 bytes..
HVL: Sending 40 bytes..
HVL: Sending 40 bytes..
HVL: Sending 40 bytes..
HVL: Sending 40 bytes..
HVL: Sending 40 bytes..
HVL: Sending 40 bytes..
HVL: Sending last 36 bytes.
HVL: #    #  ######  #       #        ####
HVL: #    #  #       #       #       #    #
HVL: ######  #####   #       #       #    #
HVL: #    #  #       #       #       #    #
HVL: #    #  #       #       #       #    #
HVL: #    #  ######  ######  ######   ####
HVL:
HVL:                                        ###
HVL: #    #   ####   #####   #       #####    ###
HVL: #    #  #    #  #    #  #       #    #   ###
HVL: #    #  #    #  #    #  #       #    #    #
HVL: # ## #  #    #  #####   #       #    #
HVL: ##  ##  #    #  #   #   #       #    #   ###
HVL: #    #   ####   #    #  ######  #####    ###
HVL:
HVL: Complete message received.
```

Text in file *msg*

```
================================================================================
                            SIMULATION STATISTICS
================================================================================
Simulation finished at time 15905

Total number of TBX clocks:                                        6830
Total number of TBX clocks spent in HDL time advancement:          3181
Total number of TBX clocks spent in HDL due to callee execution:   0
Percentage TBX clocks spent in HDL time advance:                   46.57 %
--------------------------------------------------------------------------------
Total CPU time (user mode):                                        0.02 seconds.
Total time spent:                                                  0.02 seconds.
--------------------------------------------------------------------------------
Info!     [RTS-20052]: : Freeing VeloceAVB1 license(s) ...
Info!     [RTS-20088]: : Freeing VeloceRuntime1 license ...
Info!     [TCLC-5501]: : Disconnected from emulator.
Info!     [TCLC-5501]: : project database unlocked.
Info!     [TCLC-5663]: : Shutting down the user runtime session.
```


Results/logs after running emulation (simulation)



Quiz: discuss with neighbor

- SCE-MI functions
  - What is distinction between import/export functions?
  - What is meant by 0 time functions
  - What are some uses for import/export functions?