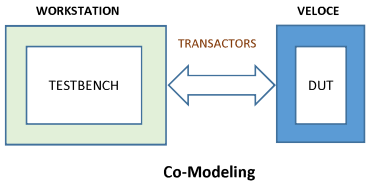
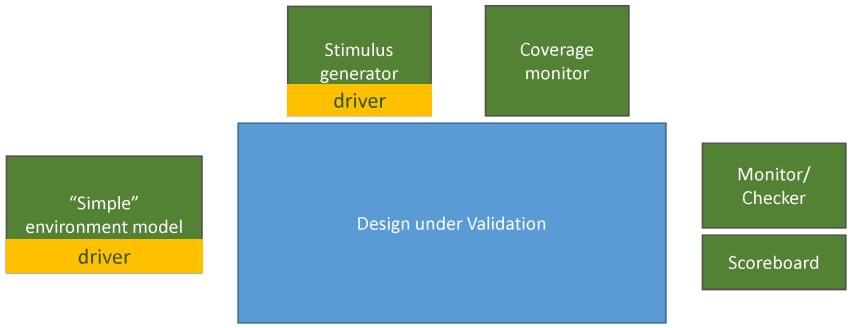
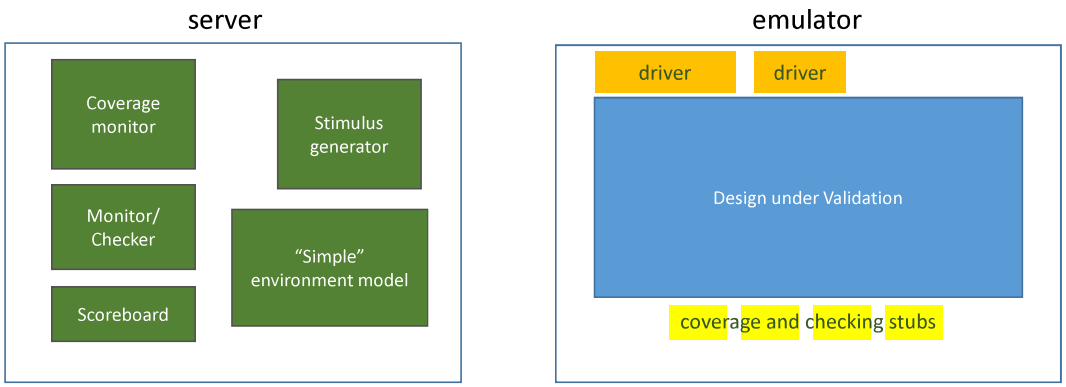
TBX

Test Bench Express

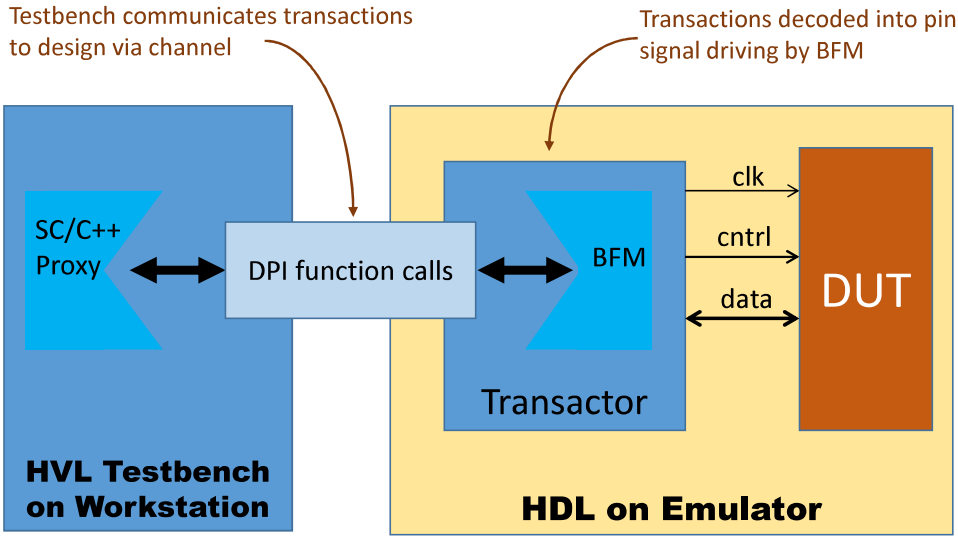


See also TBXTrainingSlides.pdf on veloce.ece.pdx.edu





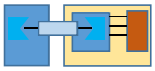
Transaction Based Methodology



Testbench in C, SV, SystemC, etc
- Includes non-synthesizable or untimed components

Synthesizable part (timed) of design running on emulator

Transactor Abstraction



- Components
  - Bus Functional Models (BFM)
    - Pin level DUT stimulus details encapsulation
    - cycle-accurate clocked events to drive DUT's signal interface
  - Channel
    - reuseable component that manages protocol between hw/sw
  - SC/C++ Proxy
    - Software models to access messages to and from the hardware side
- Extension of Testbench driver concept
  - Simplifies programming/connections
    - Testbench-HVL focuses on transaction level, un-timed testing concerns
    - Testbench-HDL focuses on driving series of cycle accurate clocked events
  - Minimize information passing between server/emulator
    - Some of the transactor runs on the workstation, some on the emulator

Mentor Graphics TBX Features

1. SCE-MI 2.0 support
   - Building blocks for transaction level communication between HVL and HDL
   - Directed Programming Interface (DPI):
   - Transaction pipes

2. Extended RTL (XRTL): Mentor Graphics proprietary
   - Added support for some behavioral RTLconstructs (makes them synthesizable)
   - Complete code compatibility and scalability between software and acceleration

SCE-MI: Standard Co-Emulation Modeling Interface

- Synthesis friendly subset of SV DPI
  - 0-time DPI functions: no simulation/emulation time "counted"/"consumed
    - import (hw→sw)
    - export (sw→hw)
  - Simple data types
    - 2-state bit vectors
    - Scalar types such as int, byte, long, etc

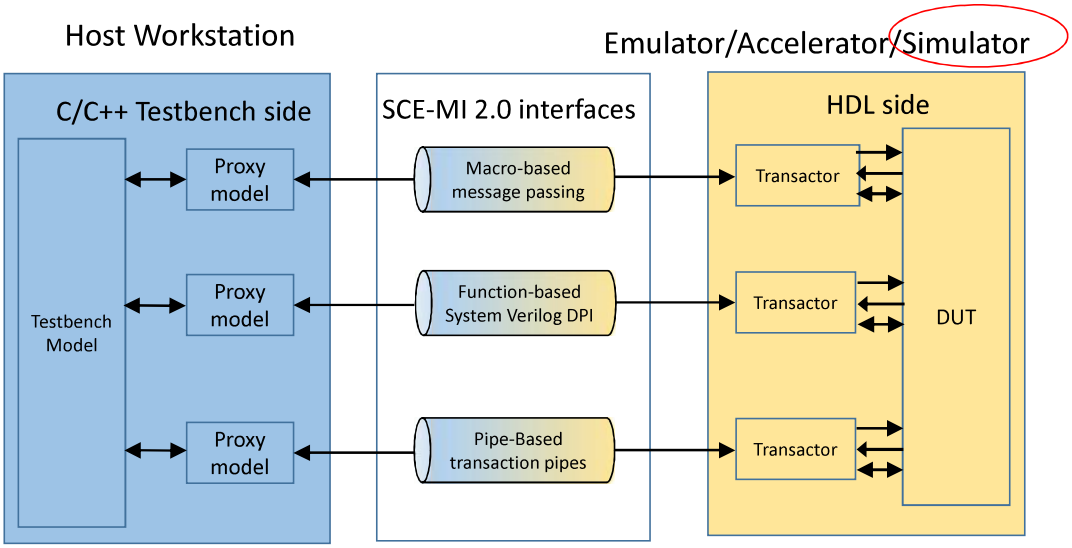
Performance bottleneck: many calls when want to send lots of data

- Transaction pipes facility for connecting models with stream-able transaction channels
  - unidirectional FIFO
  - Buffered mechanism for streaming data
    - Don't wait: data put in FIFO and test bench can continue

- SCE-MI 2.0 specifics semantics only on the interface between the hardware and software (the channel)
  - No restrictions on the modeling subset
  - Defined by Accellera. See: www.accellera.org/activities/itc

SCE-MI 2.0 Use Models

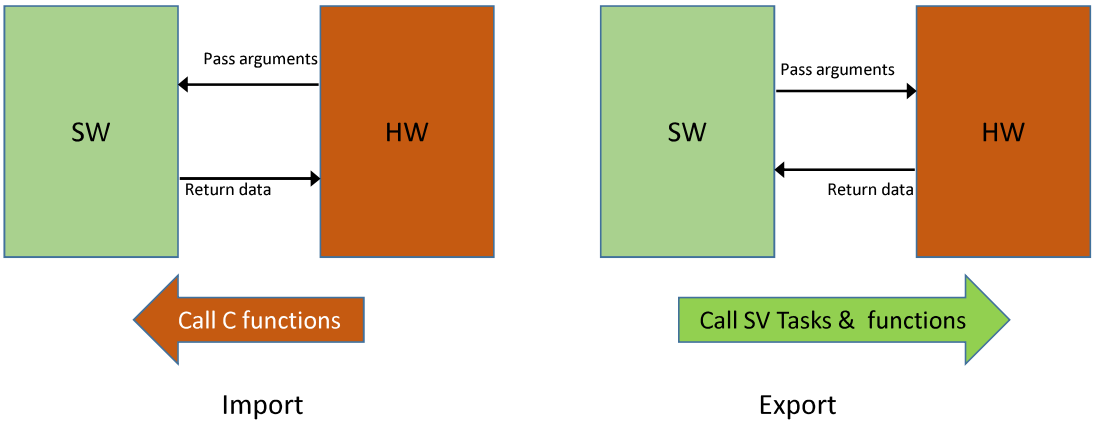
No changes required between simulation and emulation env



SCE-MI 2.0 (SV DPI interface)

DPI-C interface for C and C++

- Makes it easier to integrate HVL (C) code with SV code
- Automatic data type conversion



Function-based use model

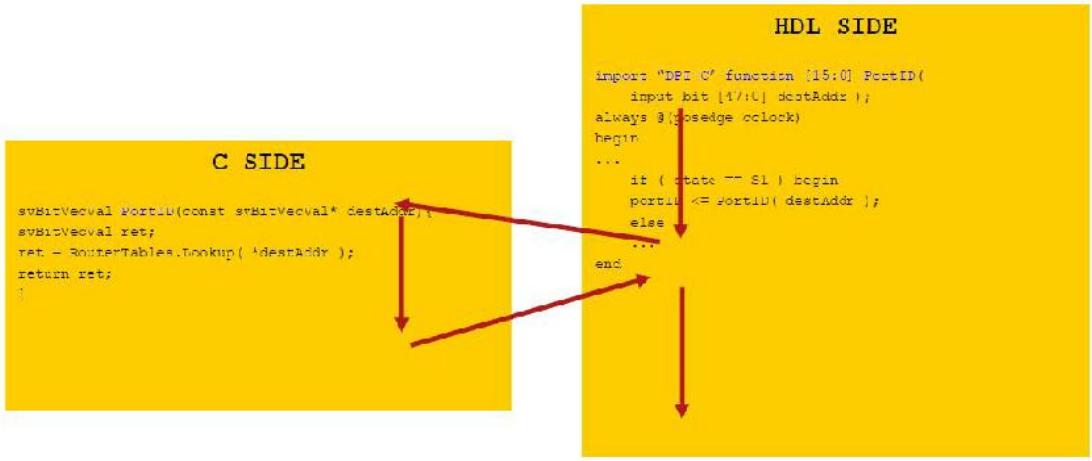
- DPI is API-less
  - Simple function calls
  - Create your own API function
- Function call itself is the transaction
  - Function call arguments are the transaction's named data members
  - Function calls can have inputs, outputs, both or neither
- Define a function in one language, call it from the other
  - Functions are defined and called in their native languages
  - The "golden principle" of DPI - on each side, the calls look and behave the same as native function calls for that language

Subset of DPI Data Type Mapping Supported in SCE-MI 2.0

DPI formal argument types	Corresponding type mapped in C
Scalar basic types: byte	Scalar basic types: char
byte unsigned	unsigned char
shortint	short int
shortint unsigned	unsigned short int
int	int
int unsigned	unsigned int
longint	long long
longint unsigned	unsigned long long
scalar values of type bit	unsigned char (with specifically defined values)
packed one-dimensional arrays of type bit and logic	canonical arrays of svBitVecVal and svLogicVecVal

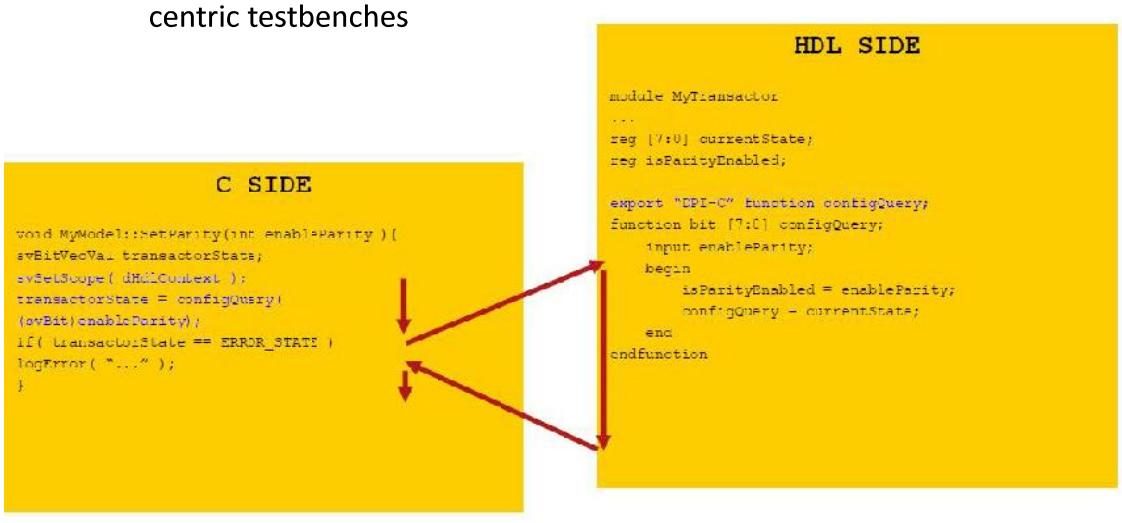
Imported 0-time DPI function call

- HDL-to-C Calls
  - A function written in C can be called from HDL
  - The C function does not consume simulated time
  - Call declaration and invocation syntax and semantics match a Verilog function
  - This is the workhorse function call type for HDL centric testbenches
  - It can also be used to model interrupts in HVL centric testbenches



Exported 0-time DPI function call

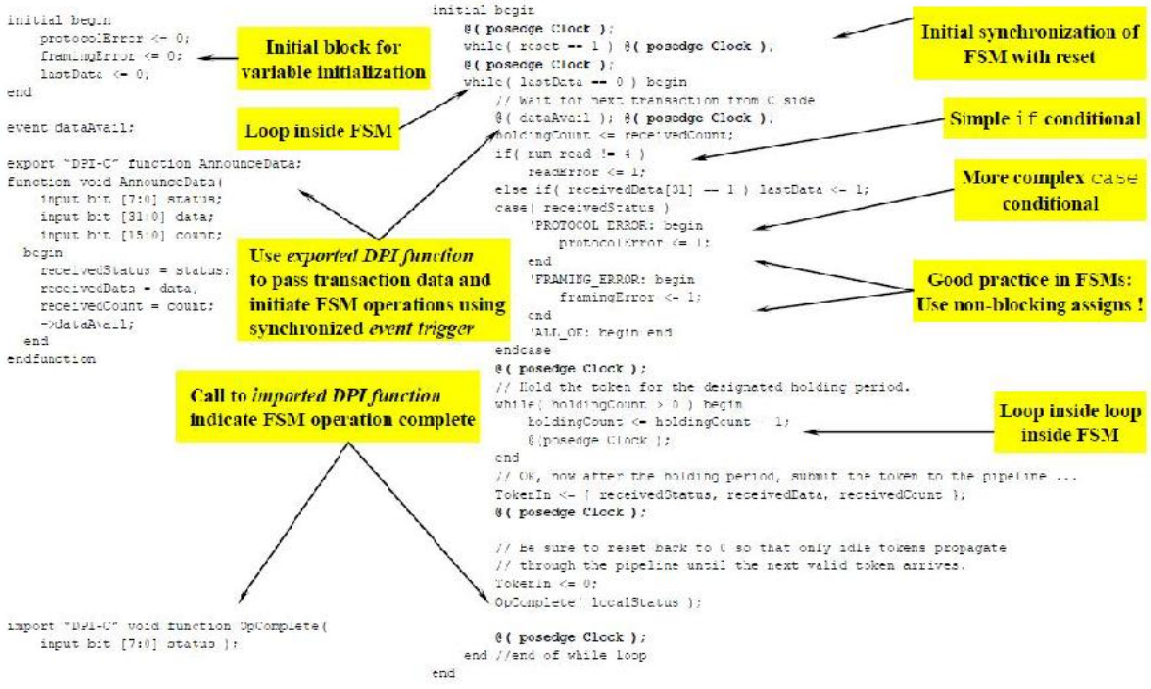
- C-to-HDL Calls
  - A function written in HDL can be called from C
  - The HDL function cannot advance time
  - Main means of doing config operations on transactors from HVL centric testbenches



What is XRTL and What it isn't

- **XRTL or Extended RTL is modeling subset of SystemVerilog**
  - Subset chosen to ensure high performance on emulator
  - RTL with a restricted set of behavioral construct extensions
  - Allows only clock synchronous RTL modeling style and constructs with a well defined set of extensions
  - Supports implicit style state machines for dramatic improvement over conventional RTL in *ease-of-modeling* of transactors
- **XRTL is not implied to be**
  - Co-simulation acceleration
  - Taking ALL behavioral verilog constructs and accelerating the simulation
  - A guarantee that what works in NCSim/simulator will work in XRTL unless coding restrictions are followed

XRTL – Anatomy of an XRTL Transactor



XRTL Transactor example observations

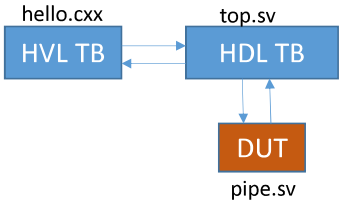
- XRTL does not support all behavioral constructs
  - But works on any simulator.
  - Functionality not changed
  - Style changed so code can be synthesized for emulator
- Initial blocks
  - Must write in certain way but doesn't change simulation semantics
    - Pragma used for clocking so can be synthesized: //tbx clkgen
- Initial blocks made synthesizable
  - Events
  - Implicit state machine (unroll explicit state machine)
- Export/import calls
- Loops, but with restrictions

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



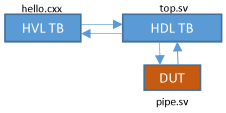
- **HDL testbench: hdl/top.sv**
  - generates clock/reset
  - fetches byte stream from HVL and pushes into DUT (pipe)
  - monitors DUT output,
    - collects it in a buffer
    - sends the outcoming bytes (maximum 40 at a time) to HVL
  - terminates using $finish when all bytes are received

- **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.

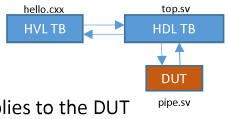
Hello.cxx

```
#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;
}
```

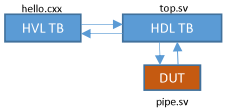
Mentor Graphics hello world example




```
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);
```

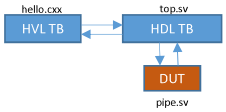
```
// XRTL FSM
// fetches the byte stream from HVL applies to the DUT
// one each clock. It stops once it receives eom.

  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
```

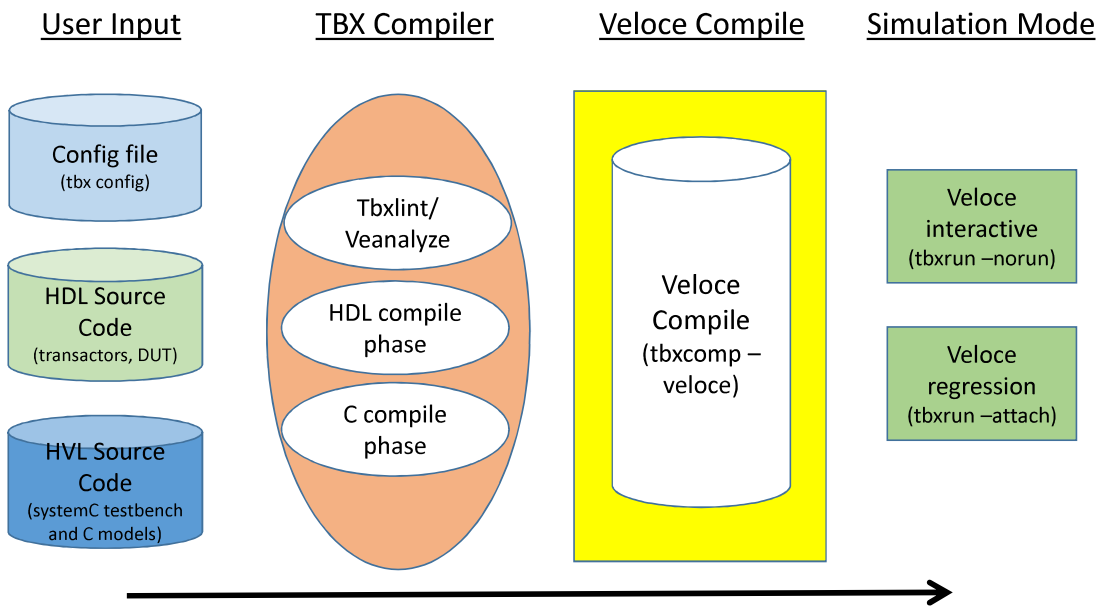
Mentor Graphics hello world example




```
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
```

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

```
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)