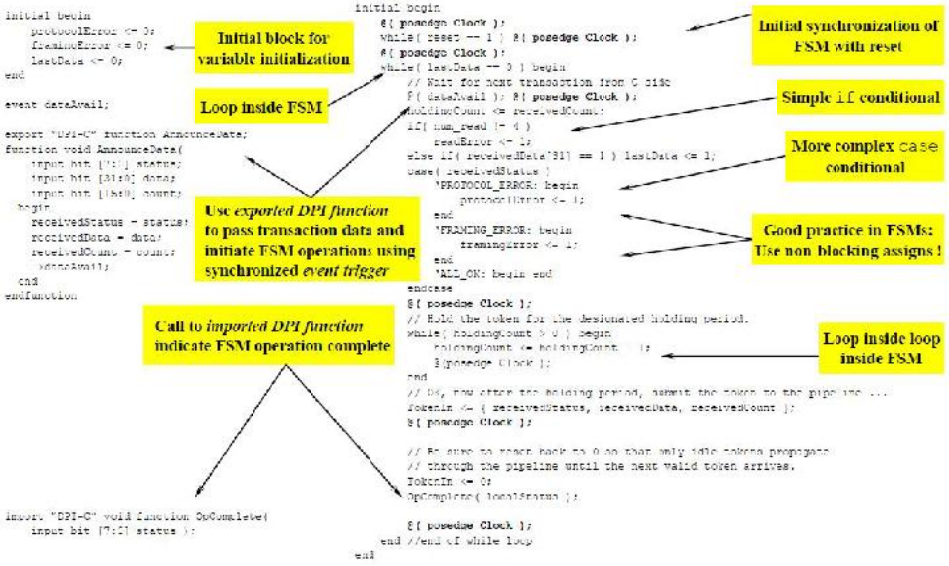
The Veloce is up!

- For setup see *D2L: Tool Resources>Emulation Information*
  - Veloce_SetupUsage_OS3_v1
  - New tutorial steps (gui steps replaced with makefile)
    - ICE_Adder_Tutorial_OS3.tgz-1
    - ICE_Legacy_Steps document (also in tgz file)
- Documentation and examples in directory: $VMW_HOME
  - (variable defined after you follow the new setup)
- If you're connected to velocesolo via VNC, you'll need to kill/restart VNC after you make the setup changes to set new path variables

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

XRTL – Anatomy of an XRTL Transactor



Anatomy of an XRTL Transactor

```
initial begin
     protocolError <=0;
     framingError <= 0;
     lastData <= 0;
end

Event dataAvail;

export "DPI-C" function AnnounceData;
function void AnnounceData(
     input bit[7:0] status;
     input bit [31:0] data;
     input bit [15:0] count;
     begin
          receivedStatus = status;
          receivedData = data;
          receivedCount = count;
          ->dataAvail
     end
end function

import "DPI-C" void function OpComplete(
          input bit [7:0] status );
```

```
initial begin
   @(posedge Clock );

   while(reset==1)@(posedge Clock);
   @(posedge Clock );

   while( lastData == 0 ) begin
       // Wait for next transaction from C side
       @( dataAvail ); @( posedge Clock );
       holdingCount <= receivedCount;
       if(num_read != 4)
              readError <=1;
       else if( receivedData[31] == 1) lastData <=1;
       case( receivedStatus )
              'PROTOCOL_ERROR: begin
                    protocolError <=1;
              end
              'FRAMING_ERROR: begin
                    framingError <=1;
              end
              'ALL_OK: begin end
       endcase
       @( posedge Clock );
       // Hold the token for the designated holding period.
       while( holdingCount > 0 ) begin
              holdingCount <= holdingCount -1;
              @(posedge Clock );
       end
       // OK, now after the holding period, submit the token to the pipeline…
       TokenIn <= {receivedStatus, receivedData, receivedCount };
       @( posedge Clock );

       // Be sure to reset back to 0 so that only idle tokens propagate
       // through the pipeline until the next valid token arrives.
       TokenIn <= 0;
       OpComplete( localStatus );

       @( posedge Clock );
    end // end of while loop
end
```

Anatomy of an XRTL Transactor

```
initial begin
     protocolError <=0;
     framingError <= 0;
     lastData <= 0;
end
```

Initial block for variable initialization

Anatomy of an XRTL Transactor

```
initial begin
     protocolError <=0;
     framingError <= 0;
     lastData <= 0;
end
```

```
initial begin
   @(posedge Clock );

   while(reset==1)@(posedge Clock);
   @(posedge Clock );
```

Initial synchronization of FSM with reset

```
end
```

Anatomy of an XRTL Transactor

```
initial begin
     protocolError <=0;
     framingError <= 0;
     lastData <= 0;
end
```

```
initial begin
   @(posedge Clock );

   while(reset==1)@(posedge Clock);
   @(posedge Clock );

   while( lastData == 0 ) begin

       @( posedge Clock );
    end // end of while loop
end
```

Initial synchronization of FSM with reset

Anatomy of an XRTL Transactor

```
initial begin
     protocolError <=0;
     framingError <= 0;
     lastData <= 0;
end

Event dataAvail;

export "DPI-C" function AnnounceData;
function void AnnounceData(
     input bit[7:0] status;
     input bit [31:0] data;
     input bit [15:0] count;
     begin
          receivedStatus = status;
          receivedData = data;
          receivedCount = count;
          ->dataAvail
     end
end function
```

```
initial begin
   @(posedge Clock );

   while(reset==1)@(posedge Clock);
   @(posedge Clock );

   while( lastData == 0 ) begin
       // Wait for next transaction from C side
       @( dataAvail ); @( posedge Clock );

       @( posedge Clock );
    end // end of while loop
end
```

Use *exported DPI function* to pass transaction data and initiate FSM operations using synchronized *event trigger*

Anatomy of an XRTL Transactor

```
initial begin
     protocolError <=0;
     framingError <= 0;
     lastData <= 0;
end

Event data Avail;

export "DPI-C" function AnnounceData;
function void AnnounceData(
     input bit[7:0] status;
     input bit [31:0] data;
     input bit [15:0] count;
     begin
          receivedStatus = status;
          receivedData = data;
          receivedCount = count;
          ->dataAvail
     end
end function

import "DPI-C" void function OpComplete(
          input bit [7:0] status );
```

```
initial begin
   @(posedge Clock );

   while(reset==1)@(posedge Clock);
   @(posedge Clock );

   while( lastData == 0 ) begin
       // Wait for next transaction from C side
       @( dataAvail ); @( posedge Clock );
```

Call to *imported DPI function* to indicate FSM operation is complete

```
       OpComplete( localStatus );

       @( posedge Clock );
    end // end of while loop
end
```


Anatomy of an XRTL Transactor

```
initial begin
     protocolError <=0;
     framingError <= 0;
     lastData <= 0;
end

Event data Avail;

export "DPI-C" function AnnounceData;
function void AnnounceData(
     input bit[7:0] status;
     input bit [31:0] data;
     input bit [15:0] count;
     begin
          receivedStatus = status;
          receivedData = data;
          receivedCount = count;
          ->dataAvail
     end
end function

import "DPI-C" void function OpComplete(
          input bit [7:0] status );
```

```
initial begin
   @(posedge Clock );

   while(reset==1)@(posedge Clock);
   @(posedge Clock );

   while( lastData == 0 ) begin
       // Wait for next transaction from C side
       @( dataAvail ); @( posedge Clock );
       holdingCount <= receivedCount;
       if(num_read != 4)
              readError <=1;
       else if( receivedData[31] == 1) lastData <=1;
```

Simple *if* conditional

```
       OpComplete( localStatus );

       @( posedge Clock );
    end // end of while loop
end
```

Anatomy of an XRTL Transactor

```
initial begin
     protocolError <=0;
     framingError <= 0;
     lastData <= 0;
end

Event data Avail;

export "DPI-C" function AnnounceData;
function void AnnounceData(
     input bit[7:0] status;
     input bit [31:0] data;
     input bit [15:0] count;
     begin
          receivedStatus = status;
          receivedData = data;
          receivedCount = count;
          ->dataAvail
     end
end function

import "DPI-C" void function OpComplete(
          input bit [7:0] status );
```

```
initial begin
   @(posedge Clock );

   while(reset==1)@(posedge Clock);
   @(posedge Clock );

   while( lastData == 0 ) begin
       // Wait for next transaction from C side
       @( dataAvail ); @( posedge Clock );
       holdingCount <= receivedCount;
       if(num_read != 4)
              readError <=1;
       else if( receivedData[31] == 1) lastData <=1;
       case( receivedStatus )
              `PROTOCOL_ERROR: begin
                    protocolError <=1;
              end
              `FRAMING_ERROR: begin
                    framingError <=1 1;
              end
              `ALL_OK: begin end
       endcase

       OpComplete( localStatus );

       @( posedge Clock );
    end // end of while loop
end
```

More complex *case* conditional

Anatomy of an XRTL Transactor

```
initial begin
     protocolError <=0;
     framingError <= 0;
     lastData <= 0;
end

Event data Avail;

export "DPI-C" function AnnounceData;
function void AnnounceData(
     input bit[7:0] status;
     input bit [31:0] data;
     input bit [15:0] count;
     begin
          receivedStatus = status;
          receivedData = data;
          receivedCount = count;
          ->dataAvail
     end
end function

import "DPI-C" void function OpComplete(
          input bit [7:0] status );
```

```
initial begin
   @(posedge Clock );

   while(reset==1)@(posedge Clock);
   @(posedge Clock );

   while( lastData == 0 ) begin
       // Wait for next transaction from C side
       @( dataAvail ); @( posedge Clock );
       holdingCount <= receivedCount;
       if(num_read != 4)
              readError <=1;
       else if( receivedData[31] == 1) lastData <=1;
       case( receivedStatus )
              `PROTOCOL_ERROR: begin
                    protocolError <=1;
              end
              `FRAMING_ERROR: begin
                    framingError <=1;
              end
              `ALL_OK: begin end
       endcase

       OpComplete( localStatus );

       @( posedge Clock );
    end // end of while loop
end
```

Good practice in FSMs: use non-blocking assigns!

Anatomy of an XRTL Transactor

```
initial begin
     protocolError <=0;
     framingError <= 0;
     lastData <= 0;
end

Event data Avail;

export "DPI-C" function AnnounceData;
function void AnnounceData(
     input bit[7:0] status;
     input bit [31:0] data;
     input bit [15:0] count;
     begin
          receivedStatus = status;
          receivedData = data;
          receivedCount = count;
          ->dataAvail
     end
end function

import "DPI-C" void function OpComplete(
          input bit [7:0] status );
```

```
initial begin
   @(posedge Clock );

   while(reset==1)@(posedge Clock);
   @(posedge Clock );

   while( lastData == 0 ) begin
       // Wait for next transaction from C side
       @( dataAvail ); @( posedge Clock );
       holdingCount <= receivedCount;
       if(num_read != 4)
              readError <=1;
       else if( receivedData[31] == 1) lastData <=1;
       case( receivedStatus )
              `PROTOCOL_ERROR: begin
                    protocolError <=1;
              end
              `FRAMING_ERROR: begin
                    framingError <=1;
              end
              `ALL_OK: begin end
       endcase
       @( posedge Clock );
       // Hold the token for the designated holding period.
       while( holdingCount > 0 ) begin
           holdingCount <= holdingCount -1;
           @(posedge Clock );
       end

       OpComplete( localStatus );

       @( posedge Clock );
    end // end of while loop
end
```

loop inside loop inside FSM

Anatomy of an XRTL Transactor

```
initial begin
     protocolError <=0;
     framingError <= 0;
     lastData <= 0;
end

Event data Avail;

export "DPI-C" function AnnounceData;
function void AnnounceData(
     input bit[7:0] status;
     input bit [31:0] data;
     input bit [15:0] count;
     begin
          receivedStatus = status;
          receivedData = data;
          receivedCount = count;
          ->dataAvail
     end
end function

import "DPI-C" void function OpComplete(
          input bit [7:0] status );
```

```
initial begin
   @(posedge Clock );

   while(reset==1)@(posedge Clock);
   @(posedge Clock );

   while( lastData == 0 ) begin
       // Wait for next transaction from C side
       @( dataAvail ); @( posedge Clock );
       holdingCount <= receivedCount;
       if(num_read != 4)
              readError <=1;
       else if( receivedData[31] == 1) lastData <=1;
       case( receivedStatus )
              `PROTOCOL_ERROR: begin
                    protocolError <=1;
              end
              `FRAMING_ERROR: begin
                    framingError <=1;
              end
              `ALL_OK: begin end
       endcase
       @( posedge Clock );
       // Hold the token for the designated holding period.
       while( holdingCount > 0 ) begin
           holdingCount <= holdingCount -1;
           @(posedge Clock );
       end
       // OK, now after the holding period, submit the token to the pipeline…
       TokenIn <= {receivedStatus, receivedData, receivedCount };
       @( posedge Clock );

       // Be sure to reset back to 0 so that only idle tokens propagate
       // through the pipeline until the next valid token arrives.
       TokenIn <= 0;
       OpComplete( localStatus );

       @( posedge Clock );
    end // end of while loop
end
```

Anatomy of an XRTL Transactor

```
initial begin
     protocolError <=0;
     framingError <= 0;
     lastData <= 0;
end

Event data Avail;

export "DPI-C" function AnnounceData;
function void AnnounceData(
     input bit[7:0] status;
     input bit [31:0] data;
     input bit [15:0] count;
     begin
          receivedStatus = status;
          receivedData = data;
          receivedCount = count;
          ->dataAvail
     end
end function

import "DPI-C" void function OpComplete(
          input bit [7:0] status );
```

```
initial begin
   @(posedge Clock );

   while(reset==1)@(posedge Clock);
   @(posedge Clock );

   while( lastData == 0 ) begin
       // Wait for next transaction from C side
       @( dataAvail ); @( posedge Clock );
       holdingCount <= receivedCount;
       if(num_read != 4)
              readError <=1;
       else if( receivedData[31] == 1) lastData <=1;
       case( receivedStatus )
              `PROTOCOL_ERROR: begin
                    protocolError <=1;
              end
              `FRAMING_ERROR: begin
                    framingError <=1;
              end
              `ALL_OK: begin end
       endcase
       @( posedge Clock );
       // Hold the token for the designated holding period.
       while( holdingCount > 0 ) begin
           holdingCount <= holdingCount -1;
           @(posedge Clock );
       end
       // OK, now after the holding period, submit the token to the pipeline…
       TokenIn <= {receivedStatus, receivedData, receivedCount };
       @( posedge Clock );

       // Be sure to reset back to 0 so that only idle tokens propagate
       // through the pipeline until the next valid token arrives.
       TokenIn <= 0;
       OpComplete( localStatus );

       @( posedge Clock );
    end // end of while loop
end
```


Anatomy of an XRTL Transactor

```
initial begin
     protocolError <=0;
     framingError <= 0;
     lastData <= 0;
end

Event dataAvail;

export "DPI-C" function AnnounceData;
function void AnnounceData(
     input bit[7:0] status;
     input bit [31:0] data;
     input bit [15:0] count;
     begin
          receivedStatus = status;
          receivedData = data;
          receivedCount = count;
          ->dataAvail
     end
end function

import "DPI-C" void function OpComplete(
          input bit [7:0] status );
```

```
initial begin
   @(posedge Clock );

   while(reset==1)@(posedge Clock);
   @(posedge Clock );

   while( lastData == 0 ) begin
       // Wait for next transaction from C side
       @( dataAvail ); @( posedge Clock );
       holdingCount <= receivedCount;
       if(num_read != 4)
              readError <=1;
       else if( receivedData[31] == 1) lastData <=1;
       case( receivedStatus )
              `PROTOCOL_ERROR: begin
                    protocolError <=1;
              end
              `FRAMING_ERROR: begin
                    framingError <=1;
              end
              `ALL_OK: begin end
       endcase
       @( posedge Clock );
       // Hold the token for the designated holding period.
       while( holdingCount > 0 ) begin
           holdingCount <= holdingCount -1;
           @(posedge Clock );
       end
       // OK, now after the holding period, submit the token to the pipeline…
       TokenIn <= {receivedStatus, receivedData, receivedCount };
       @( posedge Clock );

       // Be sure to reset back to 0 so that only idle tokens propagate
       // through the pipeline until the next valid token arrives.
       TokenIn <= 0;
       OpComplete( localStatus );

       @( posedge Clock );
    end // end of while loop
end
```

Imported/Exported Functions

Imported/Export Tasks

Imported Functions (HDL to HVL)

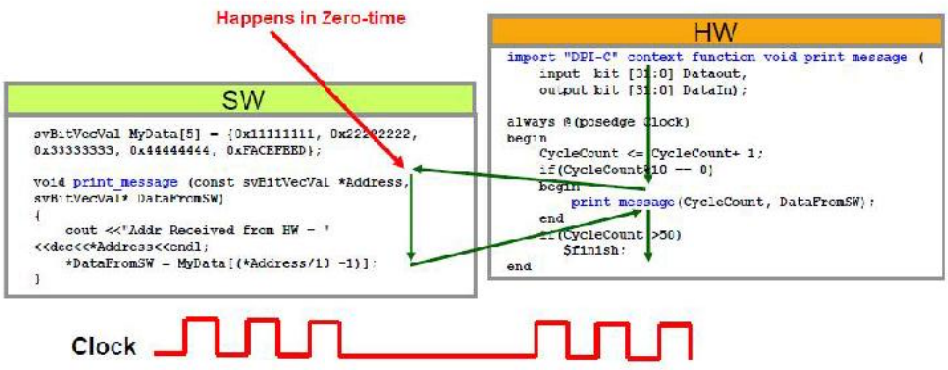
- 0 time tasks/functions initiated by the HDL-side
  - Called by the HDL transactor from a "timed procedural block"
    - HDL calling process is suspended until the call returns (like remote procedure call)
      - Apparent instantaneous amount of simulated time.
  - Defined in the HVL testbench
    - HVL side can post to 'un-timed threads'
- Uses:
  - Provide HDL with data from complex algorithm
    - Randomization, math, CRC, hash tables
  - Pass data to HVL, synchronize activity (recall co-begin, fork/join, etc)
  - File I/O
  - Interrupts

Great facility to take advantage of HVL functionality
But 0 time calls do slow down emulation
→ As much as possible, code testbench on the HDL side
→ Seek to limit the number of function calls made!

Using Imported Functions

- Calling block requirements
  - Must be RTL implicit/explicit clocked always block
  - Can not use asynchronous set/reset
  - Can call from initial block (modeling implicit SM) (XRTL special)
  - Only //tbx clkgen generated clocks can be used
- DPI call can not be made on the edge resulting from clock initialization (the first one)

Calling Imported HDL-to-C Call: print_message



Imported Function

import DPI-C [pure|context] [<c_alias> =] function [<return type>] <function name> [<attributes>] <function arguments>;

Example:

```
import DPI-C pure function bit[7:0] getData(input bit[3:0] arg1, input \
bit[3:0] arg2);
import DPI-C context function void getData(input bit[3:0] arg1, output \
bit[3:0] arg2);
import DPI-C function bit getData(input bit arg1, output int arg2);
```

TBX Supports void return type

One or more output arguments are supported

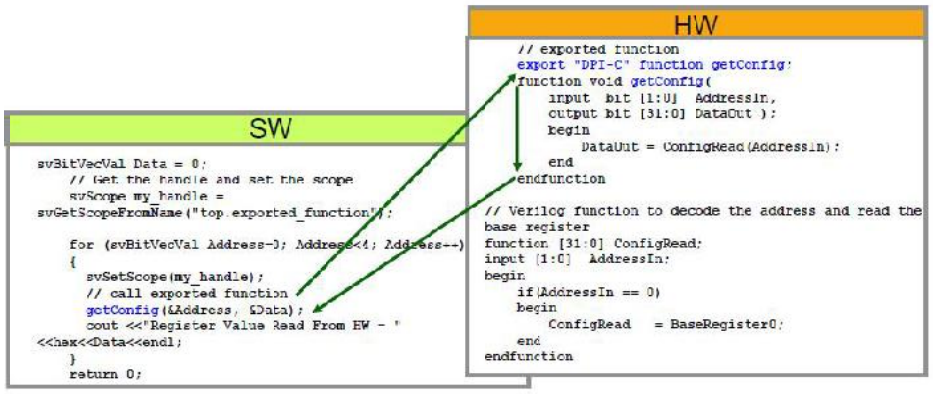
- Imported functions can only be invoked from special XRTL processes (also known as XRTL FSMs).

1 27 • TBX Vellocc Training

Exported Functions

- 0 Time function initiated by the testbench
  - Called by HVL testbench
    - HVL side is 'untimed'
    - Control transfers from thread until the function returns like RPC)
  - Defined in the HDL transactor
    - HDL side is called inside a timed model
    - Simulation time is **frozen** when the call is made
- Uses:
  - Configure a register value
  - Initial or "kick-off" FSM operation in transactor
  - Query hardware state or register value
  - Error injector

Calling Exported C-to-HDL Call: getConfig



Exported Function

export DPI-C function <function_name>

Example:

export DPI-C function getData;
export DPI-C myCName = function getData;

- The declared function must be defined in the HDL code
- Supports return type void
- Can have one or more output arguments
- Supports HDL code to call functions/tasks that have been exported

Imported Task

import "DPI-C" [context] task <task_name>

Example:

```
import "DPI-C" task getData(input bit[7:0] intr, output bit[127:0] data);
import "DPI-C" context task getData(output bit[127:0] data);
import "DPI-C" context myCName= task getData(output bit[127:0] data);
```

- TBX does not support automatic imported tasks (re-entrant) and an error is issued
- In TBX, imported tasks cannot consume time i.e. cannot invoke exported tasks that consume time
- Imported tasks can only be invoked from special XRTL processes (also known as XRTL FSMs)

Exported Task

export DPI-C task <task_name>;

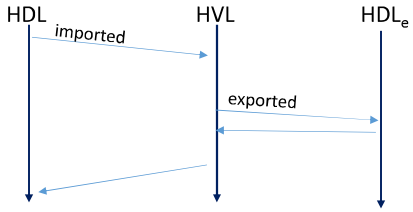
Example:

```
export DPI-C task getData;
export DPI-C myCName = task getData;
```

- TBX supports the HDL code's ability to call functions/tasks that have been exported
- Declared task must be defined in the HDL code
- TBX does not support automatic exported tasks (re-entrant) and an error is issued
- Exported tasks consume time

Nesting of DPI Functions

- TBX supports two levels of nesting
- HDL side can call an imported function that can call an exported function.



  - Once the exported function returns, it can yield control back to the imported function
- Exported function calling an imported function is not allowed

Support For Multiple Messages In 0-time

- Any number of imported function calls can be made in the same block without intervening time advancement
- Transmit multiple messages in 0-time by calling same function or multiple functions

```
always @( posedge transactor_clock ) begin
  if( reset == 1 ) begin
    // Do the reset thing ...
  else switch ( fsm_state ) begin
    case 'FSM_STATE_1: begin
      ...
      c_function1( data1, data2 );
      c_function1( data2, data3 );
      c_function2( data3, data4 );
    end
    ...
  end
  ...
end
```