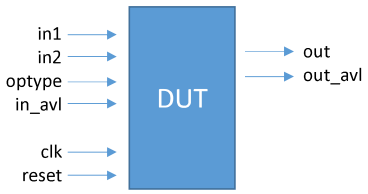
XRTL example

- HDL: simple ALU
  - add
  - subtract
  - factorial



- HVL testbench: Verilog and C
  - clock generator, reset generator, $finish
  - HDL-to-C imported calls
    - reset_completed
    - GetDataFromSoftware
    - SendDataToSoftware
    - computation_completed


```
module dut(out, out_avl,
           in1, in2, in_avl,
           optype,
           clk, reset);

// input variables
input          clk;
input          reset;
input          in_avl;
input [31:0]   in1;
input [31:0]   in2;
input [1 :0]   optype;

// output variables
output [31:0] out;
reg    [31:0] out;
output         out_avl;
reg            out_avl;

// local variables
integer        rem_val;
reg            in_process;

localparam ADD = 0;
localparam SUB = 1;
localparam FACTORIAL = 2;
localparam NONE = 3;

always @(posedge clk or negedge reset)
begin
    if (reset == 0)
    begin
        out = 0;
        out_avl = 0;
        rem_val = 0;
        in_process = 0;
    end
```

internal used to sync data fetch and execute stages

```
else  //reset != 0
    begin
        out_avl = 0;
        if (in_avl)
        begin
           out = 1;
           in_process = 1;
           rem_val = in1;
        end
        if (in_process)
        begin
           case (optype)
           ADD :
              begin
                 out = in1 + in2;
                 out_avl = 1;
                 in_process = 0;
              end
           SUB :
              begin
                 out = in1 - in2;
                 out_avl = 1;
                 in_process = 0;
              end
           FACTORIAL :
              begin
                 out = out * rem_val;
                 rem_val = rem_val - 1;
                 if (rem_val == 0)
                 begin
                    out_avl = 1;
                    in_process = 0;
                 end
              end
           endcase
        end
    end
end
endmodule
```

```
module testbench();

reg       clk;
reg       reset;
reg [1:0] optype;
event     compute;
integer   data1,data2;

//clock generator
//tbx clkgen
initial
begin
    clk = 0;
    #7; //phase delay
    forever
    begin
        clk = 1;
        #5;
        clk = 0;
        #5;
    end
end

//reset generator
//tbx clkgen
initial
begin
    reset = 0;
    #10 reset = 1;
end
```

```
//tbx clkgen
initial
begin
    #1000;
    $finish;
end

// declaration of imported tasks
import "DPI-C" task reset_completed ();
import "DPI-C" task GetDataFromSoftware (
                        output bit [31:0] data1,
                        output bit [31:0] data2,
                        output bit [1:0] opCode,
                        output bit [31:0] isMoreData);
import "DPI-C" task SendDataToSoftware (
                         input bit [31:0] data1);
import "DPI-C" task computation_completed ();


initial begin
    @(posedge clk);
    while(!reset) @(posedge clk);
    reset_completed;
end
```

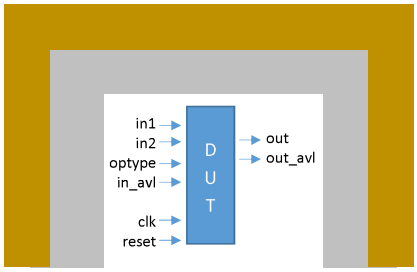


```
reg dataAvl = 0;
wire dataProcessed;
wire [31:0] data4;
integer  isMoreData ;

always @(posedge clk)
begin
    if (reset)
    begin
        GetDataFromSoftware(data1, data2, optype, isMoreData);
        dataAvl = 1;
        while (!dataProcessed)
        begin
            @(posedge clk);
            dataAvl = 0;
        end
        dataAvl = 0;
        SendDataToSoftware(data4);
        if (isMoreData == 0)
        begin
           computation_completed;
           $finish();
        end
    end
end

dut dut(data4, dataProcessed,    // outputs
        data1, data2, dataAvl,   // inputs
        optype,                  // action
        clk, reset);             // clk/reset

endmodule
```



C testbench code


```
#include <iostream>
using namespace std;

#include "tbxbindings.h"
#include "svdpi.h"
#include "stdio.h"

#define ADD 0
#define SUB 1
#define FACTORIAL 2

static int gCount = 0;
static int value1 = 0;
static int value2 = 0;
static int value_out = 0;
static int error_count = 0;
static int opcode = ADD;

int reset_completed()
{
   printf ("\n    RESET signal has been asserted ....");
   printf ("\n    Starting processing ..............\n");
   return 0;
}
```

```
int GetDataFromSoftware( svBitVecVal* data1, svBitVecVal* data2, svBitVecVal* opCode, svBitVecVal* isMoreData)
{   *isMoreData = 1;
    if (gCount < 5)
    { if ( gCount == 0 )
        {   value1 = 1200;
            value2 = 100;
            opcode = ADD;
            printf("  Testing opcode=ADD..\n");
        }
        value1 = value1 + 20 ;
        value2 = value2 + 15 ;
        printf("    -->(op1=%d, op2=%d)\n",value1,value2);
    }
    else if (gCount < 10)
    { if ( gCount == 5 )
        {
            value1 = 2200;
            value2 = 1700;
            opcode = SUB;
            printf("  Testing opcode=SUB..\n");
        }
        value1 = value1 + 20 ;
        value2 = value2 + 200 ;
        printf("    -->(op1=%d, op2=%d)\n",value1,value2);
    }
    else if (gCount == 10)
    {   opcode = FACTORIAL;
        printf("  Testing opcode=FACTORIAL..\n");
        value1 = 8;
        value2 = 1;
        printf("    -->(op1=%d, op2=%d)\n",value1,value2);
        *isMoreData = 0;
    }
  *data1  = value1;
  *data2  = value2;
  *opCode = opcode;
  gCount = gCount + 1;
  return 0;
}
```

gCount < 5	add instructions
gCount <10	sub instructions
gCount ==10	factorial instruction

```
int SendDataToSoftware( const svBitVecVal* data1)
{   value_out = *data1;
    printf("    <--(result=%d)\n",value_out);

    // Check output
    switch (opcode)
    {
    case ADD :
        if (value_out != value1 + value2)
        {   printf("=======> Mismatch (Expected: %d, Received: %d)\n", value1 + value2,value_out);
            ++error_count;
        }
        break;
    case SUB :
        if (value_out != value1 - value2)
        {   printf("=======> Mismatch (Expected: %d, Received: %d)\n", value1 - value2,value_out);
            ++error_count;
        }
        break;
    case FACTORIAL :
        if (value_out != 40320)
        {   printf("=======> Mismatch (Expected: %d, Received: %d)\n", 40320, value_out);
            ++error_count;
        }
        break;
    default :
        break;
    }
    return 0;
}


int computation_completed()
{
    if (!error_count)
        printf("All tests passed!\n");
    return 0;
}
```