AEDVICES Consulting
166B Rue du Rocher de Lorzier
38430 Moirans– France
trainings@aedvices.com
http://www.aedvices.com

# Training on
# Functional Verification Methodology
# *Using UVM*

# LAB
# Object Oriented Programming
# In SystemVerilog

---

## Objectives

Object Oriented Programming has become the programming paradigm of choice in the 1990s in the software industry. SystemVerilog, as a System Modeling extension to Verilog provides OOP constructs, such as objects and classes.

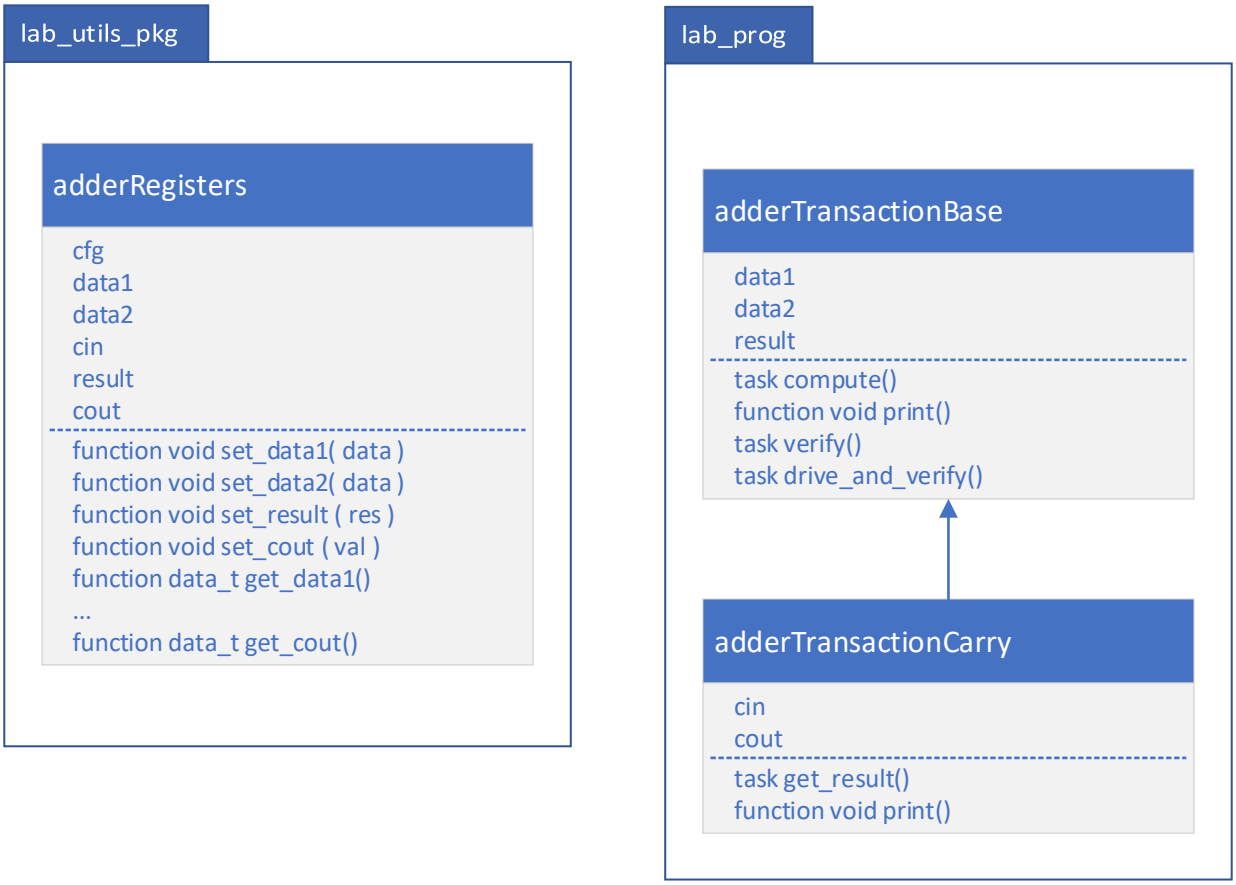
This lab goes through the main concepts of object-oriented programming using SystemVerilog language constructs. It shows how to object-oriented programming concepts are supported by SystemVerilog. The lab shows how to declare, inherit and use SystemVerilog classes as a programming language and highlight the difference between virtual and non-virtual methods.

## Global Explanation

The lab code presents three classes:

- *adderRegisters*
  Declared in the *lab_utils_pkg*
  This class represents the two DUT internal registers and provides methods to set and get them.
- *adderTransactionBase*
  Declared in the program lab_prog scope
  It has three attributes representing the three of the five DUT registers and a fourth attribute of the *adderRegister* type, it provides methods to print the attributes, to drive the DUT signals, to start the computation and to verify the DUT response.
- *adderTransactionCarry*
  This class extends the previous class, adding the remains attributes (registers from the DUT) to provide the operation with carry in and carry out.
  In addition, it overrides the print function from the parent class and has a *local* task responsible to drive the DUT to get the *result* and *cout*, role played bu the verify task in the parent class. Both classes update the *adderRegister* object.

The following UML diagram represents the classes of this lab:




## Get Started

Untar the training tarball:

```
mkdir <TRAINING_WORKAREA>
cd <TRAINING_WORKAREA>
tar zxvf <TRAINING_NAME>.tgz
```

Lab files can be found under:

<TRAINING_NAME>/labs/<LABNAME>

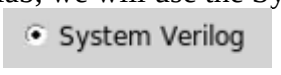
To launch the simulation,

```
cd simulation/<SIMULATOR>
./runsim.sh
```

Note: in order to launch using LSF, you must first setup the variable LAB_LAUNCHER

```
setenv LAB_LAUNCER 'bsub -I -q gui -P mcdverif -R "select[rh60]"'
```

In this lab, we will use the SystemVerilog mode. So the following checkbox should be set:




Browse and load the labs/NN-LABNAME/lab.sv file

# Instructions

The initial simulation logs should be similar to the following:

```
# ------------------------------------------------------------
# Verification Training - LAB01 - starting simulation
# ------------------------------------------------------------
#
# CASE-1
# ------------------------------------------------------------------------
# |Registers
# |data1=0x02   data2=0x03
# |result=0x05
# |cin =0xx     cout=0xx
#
# |Transaction
# |data1=0x02   data2=0x03
# |result=0x05
#
# CASE-2
# ------------------------------------------------------------------------
# |Registers
# |data1=0x07   data2=0x08
# |result=0x0f
# |cin =0xx     cout=0xx
#
# |Transaction
# |data1=0x07   data2=0x08
# |result=0x0f
#
# CASE-3
# ------------------------------------------------------------------------
# |Registers
# |data1=0x07   data2=0x08
# |result=0x0f
# |cin =0xx     cout=0xx
#
# |Transaction
# |data1=0x07   data2=0x08
# |result=0x0f
#
# CASE-4
# ------------------------------------------------------------
# Test completed
# ------------------------------------------------------------
```

## Step 1: Understand the program.

- Open the file: <SANDBOX>/labs-Xdays/labNN-systemverilog_programming /tb.sv
- Search for lab_prog
    - The user program is instantiated into the testbench
- Open the file lab_prog.sv
-
- Search for ***LAB-TODO-STEP-1***
-
- Variables a and b are of class `adderTransactionCarry`.

    Q: What are the variable members? What are their access types?
    Q: What do the *drive_and_verify* task do?

### Step 2: Add specific drive and verify functions to `adderTransactionCarry`

- In the file lab_prog.sv
- Search for ***LAB-TODO-STEP-2***
-
- Extend the adderTransactionCarry drive() function, as follow:

```
protected task drive();
    super.drive();

    addr     = 'h03;
    data_in  = this.cin;
    we       = 'h1;
    @(posedge clk);
    we       = 'h0;
    addr     = 'h03;
    @(posedge clk);
    this.addRegs.set_cin(data_out);

endtask
```

- Declare the task  verify as follow:

```
protected task verify();

    //Calcule the expected
    {this.cout, this.result} = this.addRegs.get_data1() +
this.addRegs.get_data2() + this.addRegs.get_cin();

    if({this.cout, this.result} !==
              { this.addRegs.get_cout(), this.addRegs.get_result()} ) begin
      $display("------------------------------------------------------------");
      $display("**ERROR:");
    end

endtask
```

- NOTE: The *drive* and *verify* tasks have an order to be called, we delegate the role to follow this order to the *drive_and_verify* task. This task is public, so it can be accessed by other classes and out of the class scope. Otherwise, the *drive* and *verify* tasks are protected, so they can be accessed only by the class and the children. It is important to avoid these tasks call by another source that can make mistakes and call them out of order.

- Uncomment to declare the task *drive_and_verify:*
- Compile
- Load
- Run
- Compare the differences

    - Q: What do *super.drive()* does ?
    - Q: What happens to *b.cin*, when *b.drive ()* was called with the *cin* set (as argument of the *new()* function)?
    - Q: What is the difference between the child and the parent verify task?
    - Q: What the *local* keyword before the task *get_result* means?

- Q: Why we have a different result for variable c in relation from the variable b?
- Q: Why *cin* and *cout* are not printed for variable c?
- Q: Why we get an error in the CASE-3 (Remark the difference between the register result ang the transaction result)?

### Step 3: Use virtual tasks.

- Search for ***LAB-TODO-STEP-3***
- Add "**virtual**" in front of **task print()** and **task drive()**
- Compile
- Load
- Run
  - Q: The *cin* and *cout* are printed now?
  - Q: Did you yet get an error in the CASE-3? Why?

### Step 4: Use virtual tasks.

- Search for ***LAB-TODO-STEP-4***
- Add "**virtual**" in front of **task verify()**
- Compile
- Load
- Run

  - Q: Why was the error fixed ?
  - Q: What happens if we comment the method implementation in the child class? Anything change? What?

### Step 5: Cast classes to parent class

- Search for ***LAB-TODO-STEP-5***
- Uncomment: d = c;
- Compile
  - Q: Why doesn't it compile ?
- Replace d=c with

```
assert( $cast(d,c) ) else $display("ERROR of casting");
d.drive_and_verify();
```

## Explanations

- c is a handle of adderTransactionBase
- c is assigned to b, which is an object of adderTransactionCarry
- c is casted into d, which is an object of adderTransactionCarry
- d is therefore pointing to the same object as b.
- d and b are the same.
- Virtual tasks called from c, are calling the functions of the instantiated object b, therefore of adderTransactionCarry