Code Conversion By Mapping HDL & OOPS 'Why & How'

Mr. Deepak Jain

ABSTRACT

Object Oriented Programming Structure (OOPS) has proved its importance in software development in terms of advantages like Abstraction, Encapsulation, Polymorphism, Concurrency, Modularity and Reusability. Also the Object Oriented codes are found to be more verifiable & maintainable. Hence they allow reduction in efforts for development, testing & maintenance of the software.

In current scenario, digital-VLSI design life cycle begins with modeling using some Hardware Description Language (HDL) followed by functional verification of the HDL-model by its simulation. Often, VLSI-developers show interest in getting software that simulates the functional behavior of the hardware for its analysis from different points of concern.

For the sake of effort minimization in co-designing of Digital VLSI chips and their simulating software, it is of interest to introduce automation in code conversion from HDL to OOPS and vise versa. Author's efforts in this direction are summarized in this document. The outcome of this paper may be developed as a code converter from C++ to VHDL and vise versa.

INTRODUCTION

Object Oriented Programming Structure (OOPS) has proved their utilities in terms of manageable design complexities, consistency in implementation, ease in testing, efficient maintainability of the software and increased reuse of software-modules. Object Oriented Programming Languages are quite popular in software industries for their benefits of abstraction, encapsulation, polymorphism, concurrency & reusability.

Object Oriented Design methodologies have always been good tools for fair incorporation of verifiability and maintainability in the design because object-oriented designs are inherently decoupled. In addition, object oriented systems are easier to adapt and scale i.e. larger systems can be developed by assembling reusable subsystems. [7][9]

On the other side, Digital VLSI and hardware design processes are mainly based on data-flow and control-flow techniques. In current scenario, digital-VLSI designs are first modeled in hardware description language (HDL) and simulated for functional verification. This functionally-verified HDL-model is then synthesized and the cell-placement & routing operations are performed on it, followed by timing verification. All these operations involve use of some CAD tools, e.g. Xilinx ISE (Integrated Software Environment). Wafer implementation then follows the process. [1][2][10]

Often, VLSI-developers show interest in getting software that simulates the functional behavior of the digital circuit for its analysis from different points of concern. Simulating-Software of the digital circuits and IC are getting popular also because they may be used to educate men-power about functionalities of the system, without using a real setup. Simulators are 'easy to maintain' and 'less-spacious' too. All these promote simultaneous development of hardware and their simulating software, the concept has now evolved as another branch in engineering, named Co-Designing.

Looking into the benefits of OOAD & OOPS, it is preferable to develop simulating software with Object Oriented Technology (OOT). The efforts for developing these simulators in OOPL may be minimized if some rules are defined to convert HDL-Model of a digital circuit to OOPL code directly & automatically. For this, the features of Object Orientation are required to be mapped to the concepts of HDL. This paper is about mapping of these concepts and finally coming to a conclusion in the form of a set of rules that help the developers to convert the HDL-code (VHDL specifically) to OOPL code & vise versa for a system. The outcome of this paper may work as a code-converter from VHDL to C++ and vice versa.

WHY NEED OOPS MAPPING TO VHDL

Object Orientation is basically a concept of relating a system to real world entities. An object-oriented design is implemented in an OOPL in terms of classes, objects, hierarchy and message passing. Object Oriented Programming Structure proved its utilities in terms of manageable design complexities, flexibility of implementation, ease in testing, efficient maintainability and increased reuse of system components. [3][7][8]

VHDL is one of the most popular HDLs in the industry. VHDL is an acronym for *VHSIC Hardware Description Language* where VHSIC itself is an acronym for *Very High Speed Integrated Circuits*. VHDL is a hardware description language that is used to model simple to complex digital systems. VHDL is a language for describing functional or structural behavior of a digital hardware. It is very similar to a programmin language like PASCAL but the result is a hardware modul description. It has a number of constructs that have a direc correlation with digital hardware components. It can simulat the system on different levels of abstraction from behaviora (Truth Table implementation) to structural (in terms o abstract components-instantiations). [1][4][10]

The important thing here is that VHDL is not an object oriented language and hence it may not directly be used t code for an object-oriented design. For this, a mapping i required between the Object Oriented Concepts & VHDL s that 'VHDL code for an Object Oriented Design' or 'OOP code for a given VHDL model' may be generated with the leas efforts. These features of Object Oriented Approaches a briefly described here along with their mapped relevance wit VHDL.

THE MAPPING

It is observed that VHDL also corresponds some features of Object Oriented Desig and Object Oriented Programmi Structure (OOPS) like abstractio encapsulation, reusability, polymorphism, concurrency et These concepts of sighting the 'OOPS featur correspondence' in VHDL is termed here as 'Mapping'. Th mapping may be used for conversion of program codes fro VHDL to C++ and vice versa. Abstraction, Encapsulatio Polymorphism, Reusability and Concurrency features Object-Oriented Programming Structure (OOPS) mapped VHDL as follows.

ABSTRACTION

Abstraction provides facility of hiding t internal details of a component from oth components in the system. Abstracti allows a system to be broken into sm manageable subsystems and their individual & independe development. These subsystems are finally integrated into t complete system. Abstraction property in OOPS provid description of the system in terms of different independe classes or modules, which collectively describe the syste behaviour.

VHDL provides 'abstraction' by *component* dialect th facilitates to break system behaviour into smaller subsysten The behaviour of different components can be describ independently. Finally the complete system may be model by instantiating these components in architecture of t system. There, while integrating and interconnecting t components, no component interferes in the intern structure and working of the other components, quite similar to OOPS, where integration of modules or classes has nothing to do with internal view of a class. It is only the external behavior of the module, which is of concern while integration of the system.

ENCAPSULATION

Encapsulation may be taken as stipulating the implementation of abstraction where different components may access the services of other components but none can see how the component processes for these services and also none can interfere in their internal working. Encapsulation enables selective or total information hiding in components. The concept of encapsulation in OOPS provides us the properties of inheritance and facilitates to have Public, Protected & Private accesses of data objects. [7]

In VHDL, in order to achieve encapsulation, different entities are defined and stored in library that can be used as components in architectures of other systems so that the internal information and data of a component are inaccessible (hidden) to the other components. Concept of Public, Private & Protected data objects can be visualized in VHDL as follows:

- *public data objects* of OOPL code will be represented in a VHDL model by the interfacing ports of the system i.e. the ports of the main entity of the system. It is because public data object are globally accessible, similar to the interfacing ports of some integrated circuit (IC) chip.

- *protected data objects* in a VHDL system will be represented by the ports of the subsystems or the components used in its architecture. It is so because ports of internal components can be accessed by other components within the same system, similar to the protected data objects that are accessible to the member functions only and not outside the IC.

- *private data objects* of OOP, in VHDL will be shown by the internal signals & variables defined in the VHDL architecture of the system. It is also obvious as per their accessibility within the component similar to private data objects in OOPS, accessibility within class or the function in which they are defined.

Keeping these points in concern, following six C++ mapped references may be defined for VHDL constructs:

i. Any *component* used in structural VHDL architecture (i.e. a component port-mapped with some signals) refers to an OOPS class inherited by the class referred by the main entity in VHDL.

ii. *Ports* of the VHDL main entity refer to *global data-variables* in OOPS.

iii. *Behavior of the component* used by main entity is put as '*protected functions*' in its equivalent OOPS class.

iv. *Port Mapping* is implemented as *Public Functions* of OOPS.

v. *'bit'* type of VHDL model may be referred to *'bool'* or an enumerated type may be defined for this in OOPS.

vi. Activities of assigning signal values to the input ports is implemented in '*void main()*'.

MODULARITY & REUSABILITY

Modularity defines the units of reuse. The system is divided into a number of components and these components may be reused in different parts of the system. Modularity seems to be closely related to abstraction but here the focus of application is different. In abstraction, the focus of division of system into subsystems is to hide the internal details of subsystems whereas in modularity, focus is to divide the system in reusable units called *modules* that can be compiled individually and stored in library. [14,17]

In the hardware systems, the concept of reusability may be of two types:

- One: To use the single component at different places.
- Two: To use the clones of a component at different places

For the first case, i.e. to use the single component at different places, we can use a structure as shown in *Figure 1*. In this figure, 'IW' is the width of Input port and 'OW' is the width of output port of the reusable component. The reusable component receives input through input buses I0, I1, I2 and I3 and provides output on output buses O0, O1, O2 and O3. The input buses (I0 to I3) are of width IW each and output buses (O0 to O3) are of OW each. This system can easily be simulated in VHDL using MUX, DMUX and Reusable components in library and the components needing access to the reusable component can access it through its I/O buses.

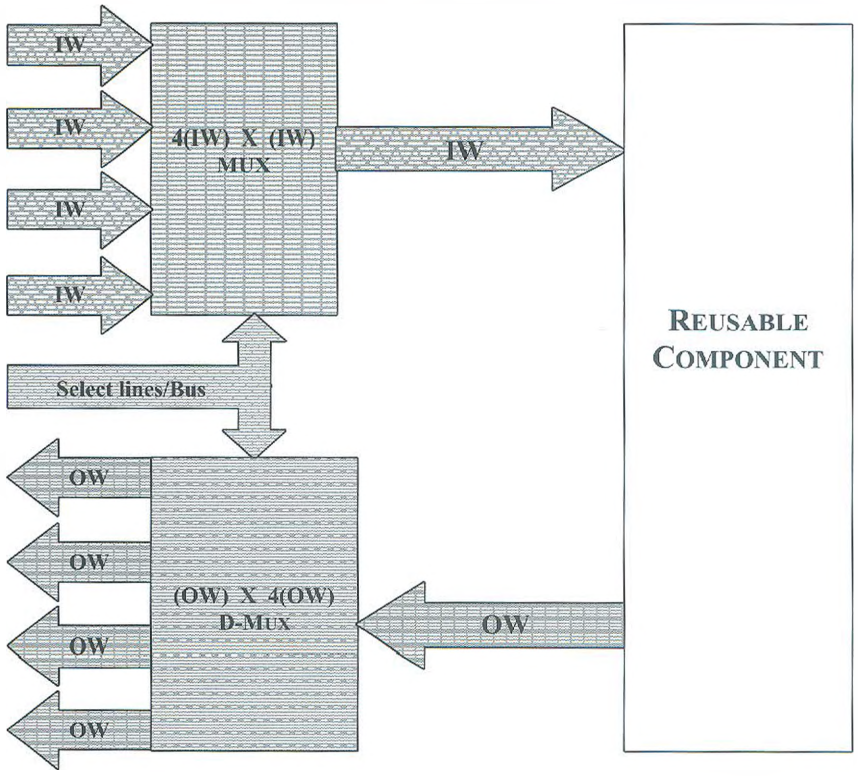

Figure 1 : Setup for reusing a component at different places

For the second case i.e. to use the clones of a smaller system at different places in a large architecture it can be defined as an entity in the library and its instantiations can be used in the architecture of the bigger system. VHDL keyword '*component*' can be used for instantiating a clone of the subsystem and '*port map*' can be used to make interconnections.

To instantiate multiple orderly-interconnected clones in VHDL, '*generate*' keyword can be used. In any OOPL, it is analogous to define multiple objects of a class & then use their functionality & behavior individually. Each of these objects will have its own identity & state but their behaviour will be same for same conditions. Similarly in VHDL, different component-instantiations have their own identity & state but all components have same behaviour as that of their base identity from which they were generated.

POLYMORPHISM

Another feature of the object-oriented concepts is to provide the facility of function overloading in terms of polymorphism. In fact, polymorphism may be defined as the capability of existence in more than one form. [8]

The similar concept in VHDL may be implemented using the subprogram overloading. In this we may have two or mor programs with same names. For example:

- function COUNT (ORANGES: INTEGER) retur INTEGER;
- function COUNT (APPLES: BIT) return INTEGER;

Here the function COUNT is said to be 'overloaded'. When call is made to either of them, it is possible to identify the righ function, which the call is actually meant for. The targe function is identified by the type of actuals (parameters passed because the two functions have different paramete types. [1] [5]

CONCURRENCY

In the concerned domain, concurrenc means to concurrent running of th functions or processes. OO designs allo independent processes to ru simultaneously in parallel. Object Oriente Program codes define the concurrent states of objects an subsystems along with their concurrent state-transition usin multithreading facilities. It is a type of explicit parallelisr dialect provided in object oriented programming languag like JAVA. Using such parallelism dialects, concurren behavior of a design can be implemented.

VHDL implements concurrent states & behavior of subsystems using "Process" statement. The statements under the "Process" block execute sequentially but the "Process" statement itself is a concurrent statement. It is similar to that the statements under a 'thread' execute sequentially but multiple threads run parallel. Multiple OOPL threads are analogous to multiple VHDL process blocks, which will finally be executed concurrently. Data flow architecture modeling is another way to achieve concurrency in VHDL models.

IDEA OF APPLYING AND USAGE OF MAPPING

The concepts of mapping of OO features to VHDL features, as discusses herein this paper, may be used to convert a given VHDL simulation model of some digital circuit to OOPL software simulation of the system and vice versa. A simple example of such application is discussed here as follows. Here given (see *Figure 2(a)*) the VHDL model of a simple Half Adder module. Using the rules of conversion as discussed, an abstract OOPL (C++) code may be generated as in *Figure 2(d)*.

VHDL Structural Model for Half Adder

```
library IEEE;
use IEEE.std_logic_1164.all;
entity adder
   port (X, Y:in bit; S, C:out bit);
end adder;

architecture adder_struct of adder is
    component XOR1
      port(XI1, XI2:in bit; XO:out bit)
    end component;

    component AND1
      port(AI1, AI2:in bit; AO:out bit)
    end component;

Begin
   HA1: XOR1 port map (X, Y, S);
   HA2: AND1 port map (X, Y, C);
End adder_struct
```

Figure 2(a): Half Adder VHDL Model

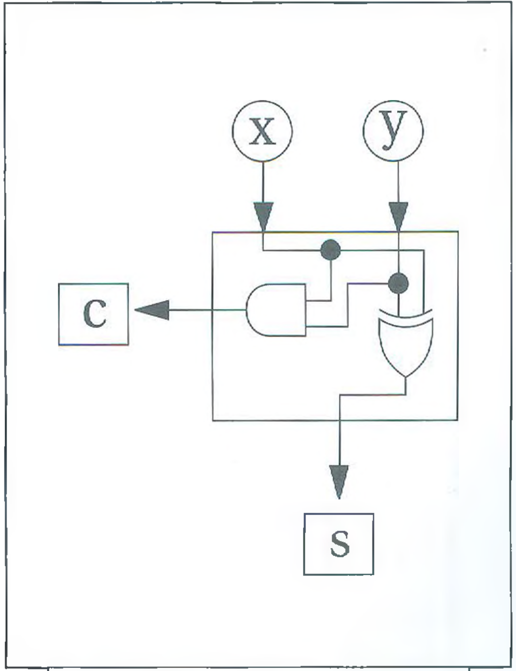

Figure 2(b): Half Adder RTL view

Rules to be followed for conversion

i. Any *component* used in structural VHDL architecture (a component port-mapped with some signals) refers to an OOPS class inherited by the class referred by the main entity in VHDL.

ii. *Ports* of the VHDL main entity refer to *global data-variables* in OOPS.

iii. *Behavior of the component* used by main entity is put as '*protected functions*' in its equivalent OOPS class.

iv. *Port Mapping* is implemented as *Public Functions* of OOPS.

v. *'bit'* type of VHDL model may be referred to *'bool'* or an enumerated type may be defined for this in OOPS.

vi. Activities of assigning signal values to the input (port is implemented in void main).

Figure 2(c): Rules of Mapping

```
#include<iostream.h>

struct input
{
  boolX;
  boolY;
}inPort;

struct output
{
  bool S;
  bool C;
}outPort;

class adder
{
  private:

  protected:
         bool XOR1(bool XI1, bool XI2)
         {
           bool ret;
             /*Here, the functionality of the XOR1 component may be implemented as described
               in its architecture*/

           return ret;
         }

         bool AND1(bool AI1, bool AI2)
         {
           bool ret;
             /*Here, the functionality of the AND1 component may be implemented as described
               in its architecture*/

           return ret;
         }

  public:
         void HA1(bool I1, bool I2)
           /*here the component used is XOR1, and hence its input port (bit, bit) can directly be
             mapped to the input parameters of the void HA1*/
         {
           outPort.S = XOR1(I1, I2);
         }

         void HA2(bool I1, bool I2)
           /*here the component used is AND1, and hence its input port (bit, bit) can directly be
             mapped to the input parameters of the void HA2*/
         {
           outPort.S = AND1(I1, I2);
         }
}
void main()
{
/*here, the values may be assigned to the input ports "inpot.X ans input.Y" as according to the
used signals by forcing values in anyVHDL simulator while simulating the application*/

  adder a1;
  a1.HA1(inport.X, inPort.Y);
  a1.HA2(inport.X, inPort.Y);
}
```

Figure 2(d): Half Adder C++ Simulation Program

CONCLUSION & FUTURE WORK

Conceptually, constructs of an object-oriented design have direct resemblance with those of VHDL models. VHDL models also exhibit the properties of object-oriented designs including abstraction, modularity, reusability, polymorphism and concurrency. To bring these concepts to a mathematical level, efforts are required to define a more formal logical platform where these concepts may be transformed into materialistic results for the benefits of VLSI and software designers. This analysis can be concluded as an effort to answers the following summarized queries:

- With this analysis, can the hardware systems be classified in terms of complexity, nature and/or some other characteristics where the Object Oriented concepts give better design results in lesser efforts?
- Can a format or concept be defined where using object-oriented language; simulation software might be developed for a given HDL model i.e. can this analysis be translated into a VHDL to OOPS code converter?
- Can a concept be defined where an HDL Model may be developed for a system from Object-Oriented Language code of its simulator i.e. can a code converter be developed to convert OOPS (C++) code to HDL (VHDL) model?
- Can a system or some concept be produced with which one may be able to implement an object-oriented design of some hardware system into its HDL code and vice versa?

REFERENCES

1 Douglas L. Perry (2000), VHDL Designing, PHI Publication.

2 Douglas L. Perry (2002), VHDL: Programming by Example, McGraw-Hill Company, New York.

3 Grady Booch (1994), Object Oriented Analysis & Design with Application, 2nd Edition, Addison-Wesley Publication.

4 James Rumbaugh (2002), Object Oriented Analysis & Design, Prentice Hall.

5 Jayaram Bhasker (1999), VHDL Primer, 3rd Edition, Prentice Hall of India.

6 Kai Hwang (2001), Advanced Computer Architecture-Parallelism, Scalability, Programmability, Tata McGraw Hill.

7 Richard C. Lee, William M. Tepfenhart (2005), UML and C++, A Practical Guide to Object Oriented Development, 2nd Edition, Prentice-Hall of India.

8 Robert Lafore (2003), Object Oriented Programming in Turbo C++, 3rd Edition, Galgotia Publication.

9 RS Pressman (1992), Software Engineering, A Practitioner's Approach, 3rd Edition, McGraw Hill Company, New York.

10 Z. Navabi (1998), VHDL: Analysis and Modeling of Digital System, 2nd Edition, McGraw Hill Company, New York