VLSI Integrated Circuit Design Representation in an Object-Oriented CAD Environment

Wlodzimierz Wróka  
Politechnika Łódzka  
Wydział Budowy Maszyn  
Bielako-Biała  
Poland

Adam Pawlak  
G.M.D.  
Project E.I.S.  
5205 St. Augustin  
F.R.Germany

Concepts and some implementation results of a design environment based on an object-oriented design data model are presented in the paper. This model has been used for the definition of the data scheme and the data manipulation language. The former has been defined in terms of VHDL and EDIF mechanisms. Both, the data scheme and the data manipulation language have been implemented in SMALLTALK. Finally, the results of experiments justifying the approach are presented.

Keywords: VLSI design environment, object-oriented programming, VHDL, SMALLTALK.

1 INTRODUCTION

Increasing complexity of integrated circuits results among others in a huge amount of design data representing various design attributes. A need for new tools processing, managing and storing so large volume of data becomes urgent. An environment integrating different CAD tools in a unified way, based on accepted international standards provides the only reasonable strategy to cope with complexities of current VLSI circuits. Although different ideas for organization and implementation of a design environment are proposed, the one focusing more and more attention is the object-oriented style.

The paper presents concepts and some implementation results of a design environment based on an object-oriented design data model. Experiences from software engineering point to object-oriented programming languages as efficient means for organization and representation of design data. The characteristic features of object-oriented programming languages, namely: inheritance, definition of classes (ADTs), encapsulation and polymorphism have been consequently utilized in the definition of our design data model. Based on this model, the data scheme and the data manipulation language will be presented. The former has been defined in terms of VHDL and EDIF mechanisms. Both, the data scheme and the data manipulation language have been implemented in SMALLTALK as a part of the object-oriented design environment (ODE). Finally, some results indicating to the advantages of our approach are provided.

2 Short overview of object-oriented CAD systems

Cheshire, GDT, STEM, Palladio, ODICE and MoDL are the examples of object-oriented CAD systems known from the literature. The first one [4] is an experimental design environment implemented in the MAINSAIL language on SUN-3/160. GDT (Generator Development Tools) [2] coming from Silicon Compilers is a UNIX-based system supporting mainly design of module generators. The STEM system (SmallTalk Environment of Module design) [5] is based on SMALLTALK and implemented on Sun3/260. Palladio [1] was the first system to combine object-oriented approach with AI techniques. The system was implemented on Xerox 1100 in LOOPS and MRS. ODICE (Object-Oriented Hardware Description In CAD Environment) [7] is an object-oriented hardware description language (HDL) based on the multilevel HDL DACAPO-III. The system MoDL (Modelling and Design Language) [11] is implemented in Lisp extended with object-oriented features (version called TULISP).

The above mentioned systems allow for structured design of integrated circuits by supporting both top-down and bottom-up design strategies. In GDT, STEM and Palladio design steps are verified through simulation. Palladio allows also for exceptions from structured design rules by introducing a concept of a partly structured design. Practice shows, that allowance for designer’s "non-systematic" interrogations may lead to a higher quality designs. A detailed design of some subcircuits prior to an initial design of other parts, is quite often simply required. All these systems have one common feature, namely a data model based on a notion of a class used to represent a digital circuit. Every instance of this digital circuit is then modelled as an object of the class.

A design data model is one of the main aspects of a design environment having a predominant influence on speci-
3 Object-oriented data model

According to [12,13] data models may be divided into record models and semantic models. Traditional models, i.e., hierarchical, network and relational, which are based on a record, don’t suit the requirements for data representation in modern CAD environments for VLSI design. They do not allow for building complex objects with complex attributes. These must be delegated to application programs. Additionally, clear definition of data consistency is missing as well as a possibility for flexible handling of different representations, versions and configurations of design data. Record structures are not useful for modelling non-formatted information, like graphs, documents, messages or programs [8].

Literature points to a new type of data models, so called semantic models [8,13]. Semantic data models provide a rich set of relationships better suited for representation of data semantics and their logical structures, enabling users to easier comprehend and employ data. One of the representatives of this class of models is just the object-oriented data model.

The object-oriented data model introduces first of all, a unified view of a system to be modelled, by requiring all elements of a modelled world to be objects. Objects in a natural way represent ideas, elements and activities of a design environment. The object-oriented data model may constitute a basis for a complete integration of a design environment, since “tools and data” are amalgamated through a data model. This is specially true for CAD environments, which are integrated on a basis of common design data stored in a data-base. A whole environment may be modelled as a complex object, whose elements like hardware, system software, CAD tools, user interface and design data are represented as objects. This enables a unified access to CAD system resources. Mapping of objects into an internal format in a data base is not visible for a user. It is thus easier to work with such a system, and a programming task constitutes a smaller burden to an implementing engineer.

The model is based on three basic relations: classification, aggregation and generalization. These can be easily illustrated on a simple example of a JK flip-flop composed from two r-s latches (Figure 1).

![Figure 1. A JK flip-flop representation based on an object-oriented data model.](image)

A relation of classification represents interdependencies between an object and a type (a.kind.of). Aggregation creates a new object from a set of other objects (a.part.of). Generalization represents a relationship between a subtype and supertype (is.a).

Two additional semantic forms convertibility and relatability may be assigned to an object-oriented data model. Convertibility means that each object is characterized through its own attribute values. Relatability points to that object’s attribute, which has a name identical with a name of the object’s type.

The above listed features of the object-oriented data model imply characteristics of the data scheme of this model. An object-oriented data model is closely related to a data manipulation language, which separates a user from an internal data organization. It offers mechanisms for data selection, extension, and modification. Data modification can be attained through operations of insertion, deletion, and update. Objects may be created, updated, and instantiated as components of more complex objects.

The object-oriented data model constitutes thus a strong means for representation of complex design data, supporting integration of an environment and simplifying data management [13]. Based on the definition of the data model, we can define a data scheme and a data manipulation language, which implemented in the SMALLTALK environment present a number of interesting aspects.

4 Design data scheme

A design data scheme for the ODE environment has been defined based on a standard hardware description language VHDL and a design exchange format EDIF. EDIF as a basis for a data scheme has been also used in [6].

Physical aspects of a design description have been attached to a structural part of the ODE data scheme. The data scheme has also global features (meta data) related to a data management system of ODE.
4.1 Basics of the ODE design data scheme

Construction of complex data types required a predefined structure of a set of primitive data types. For example, floorplanning and layout design take advantage of two primitive types: rectangle and point, which enable construction of a geometric scheme. Additional primitives like lists, collections and dictionaries should be available.

Primitive types are used to define attributes of ODE data scheme types. Among those types, the EntityElement type plays a particular role as a root super-type for all other ODE types.

```
type: EntityElement
subtype: Object
attributes: Id, BelongTo
```

The Id attribute points to an identifier of a selector element of a type, whereas the attribute BelongTo specifies a membership hierarchy. An example may be drawn from an element belonging to a library. One can also add other attributes or a list of attributes, to an EntityElement type definition. Those attributes are then inherited by all data scheme types.

```
Types containing additionally a list of declarations are subtypes of a DesignObject type.

type: DesignObject
subtype: EntityElement
attributes: DeclarationList, Status
```

The Status attribute enables a control of objects utilization. A designer’s name and a last operation on the object are simple examples of the status elements. Subtypes of the DesignObject type, among others: Procedure, Function, Library and Package comprise conceptual data supporting declaration, modelling, and aggregation of various design objects. Those objects must not be in any way related to a specific hardware object.

The DesignObject type may constitute a super-type for types representing an interface and a body of an IC design (DesignEntity, and BodyUnit).

4.2 Design entity - representation of a hardware unit

Hardware units of arbitrary complexity, called design entities are composed of two conceptual elements, namely their interface and a set of different bodies. An interface comprises among others a list of parameters and a graphical symbol. A body constitutes a description from different specification domains, e.g. behavioral, structural and/or physical. In every specification domain a number of various representations on different abstraction levels may exist.

A type DesignEntity constitutes a template for digital hardware units. Its attributes SchemeSymbol and PhysicalSymbolList contain data for a structural floorplan and a layout scheme. The other attributes PortList and PermutablePortList represent a typical set of hardware unit terminals (pins).

A DesignEntity type includes a list of formal parameter values, each determining a particular features of a design unit (attribute ParameterList). It contains also a dictionary of bodies of design entities (attribute VersionDictionary). The State attribute determines a unit’s state during a design process.

A design entity of a Cnt_Cell counter may be represented in two forms: as a textual description in VHDL and as a set of attribute values of a respective data scheme (Figure 2).

A parameter N in this example determines a placement of a respective cell in a string of cells forming a multi-bit counter.

```
library Work:
entity Cnt_Cell is
  generic (R: NATURAL := 0);
  port (Clk, Cinbar, Phil, Phi2: IN BIT;
    Cout, Outbar: OUT BIT;
    VDD, GND: Linkage PWR);
  constant Tds: TIME := 1 ns;
  Iddiv: TIME := 15 ns;
begin
  end Cnt_Cell;
```

(a)

```
```

(b)

```
```

(c)

charted DesignEntity
id: 'Cnt_Cell': library: 'Work'
declarationList: <Constant (Td, Tdiv)>
status: Status (...) shapeSymbol: Shape (...) parameterList: <N>
portList: <Clk, Cinbar, Phi, Out>
versionDictionary: <...
state: State (...)
```

Figure 2. The Cnt_Cell representation: a) a graphical symbol  b) the VHDL description  c) attribute values of a data scheme.

Every design entity contains a number of its bodies. Different types of design entity bodies are interrelated with ImplementedVersions and ImplementationOf attributes. They enable to represent a relationship between different bodies. For example, a number of various structures may be related with one functional body.

Different bodies of a design entity form a prefix of a BodyUnit type (Figure 3). A BodyUnit type is a super-type of a Behaviour type, which consecutively is inherited through a Structure type, whereas the last one is inherited by a Structure and Behaviour type. The BodyUnit type is also a super-type of the PureGeometry type representing selected aspects of a physical domain.
4.3 Functional body representation

A functional body comprises a ComponentList attribute. This attribute points to the functional components representing a behavior of a design entity. The components are Process, Block, and Concurrent Signal Assignment. They are interconnected with the entity's part through signals. Therefore, a functional body may be visualized as a graph, vertices of which represent signals and/or components. Figure 4 presents a graph representing a functional body of the Cnt.Cell counter and its VHDL description.

architecture DataFlow of Cnt_Cell is
signal state1, state2: BIT;
begin
  Cout <= Cin after (2*N+1)*Tds when state1 = '1'
  else '0';
  Coutbar <= Cinbar after (2*N+1)*Tds
  when state1 = '1'
  else '1';
  Ph2: block (Phi2 = '1')
  begin
    state2 <= guarded Cin after (4*N+6)*Tds
    when state1 = '1'
    else Cinbar after (4*N+6)*Tds;
    end block;
  Ph1: block (Phi1 = '1')
  begin
    state1 <= guarded not state2 after Tds*Tdinv;
    end block;
    Sout <= not state2 after Tdinv;
    end DataFlow;
Figure 4a. An example of a Cnt.Cell counter representation.

Vertices of the graph are connected with obliquely specified directed arcs. The last represent a direction of an information flow. Arcs directed to vertices indicating components represent signals in those components, whereas arcs directed to vertices indicating signals point to a component, which changes its signal's value.

A Process constitutes a basic component of a functional body. A list of signals activating this process is one of its attributes. A Process may also be represented as a graph consisting of the following elements:

Figure 5. An example of a process: a) a VHDL text b) a graph.

Continuous lines represent data flow, whereas intermittent lines provide control for events sequencing.
4.4 A structural body representation

A Structure type is a subtype of a Behaviour type. It inherits among others the attribute ComponentList, which in this case represents a list of structural components. Structural components are objects of the Component type. The last one represents designs being a part of a design entity structure. The Component type includes attributes: GenericType, and the ActualParameterList. The former points to an object of a declared design entity. A design structure constitutes a hierarchy of structural components, with roots on a primitive design entities level.

Figure 6. A logical scheme of a four-bit Cnt counter.

Structural features of a four-bit counter composed of four Cnt Cell counters (Figure 6) are represented as objects of data scheme types (Figure 7).

a)

architecture Str-1 of N_bit_Cnt is
component Cnt_Cell
  generic (N: NATURAL);
  port (Cin: BIT; Cout: Coutbar: out BIT;
     VDD, GND: linkage PWR);
  signal C: BIT_VECTOR (0 to N-1);
  signal Coutbar: BIT_VECTOR (0 to N-1);
begin
  for i in 0 to N generate
    if i = 0 generate
      Cnt(i): Cnt_Cell generic map (i);
      port map (Cin, Cinbar, Phil, Phi2, Q(i),
      Cz(i), Czbar(i), VDD, GND);
      end generate;
    if i > 0 and i < N generate
      Cnt(i): Cnt_Cell generic map (i);
      port map (Cz(i-1), Czbar(i-1), Phil, Phi2,
      Q(i), Cz(i), Czbar(i), VDD, GND);
      end generate;
    if i = N generate
      Cnt(i): Cnt_Cell generic map (i);
      port map (Gz(i-1), Gzbar(i-1), Phil, Phi2,
      Q(i), open, open, VDD, GND);
      end generate;
  end for_str-1;

4.5 A physical body representation

A concept for a physical domain representation is based on an assumption, that all elements of a floorplan and layout have properties of a design entity's structure. These properties may be defined from architectural to logic levels of design abstraction, since based on structural components. On a circuit level, elements of a structure are represented through electrical properties of a circuit like, transistors, resistors etc. They have explicit physical features. A declaration of a physical description is thus possible with primitive units. Physical features of a complex entity result from their structural hierarchy.

Two kinds of layout definitions, namely: a symbolic layout and a pure geometry are interrelated with their respective structural representations. They may be declared as a list of interconnected terminals, segments and contacts in the following cases: in an interface as a symbolic shape, in ports definition as a list of terminals and in components interconnections (wires).

5 Data manipulation language

This section will shortly discuss some aspects of the data manipulation language used in the ODE environment. The language provides operations, which may act on design data, i.e. data created according to the data scheme. Those are...
the operations of extension, modification and selection. The last one is perhaps the basic one in the manipulation language of ODE.

**Type: identifier**

**its: attributes**

**where: condition**

Realization of the selection operation utilizes mainly a relation of classification. Firstly, objects of a given type are noticed (a kind of). Then, a condition for this operation is verified with a grouping relation (a part of). For objects of a given type, a list of their attributes is looked for. Values of attributes are compared with those of a declared condition. Only objects fulfilling this condition are selected. A definition of selection operation must be extended with a check for inclusion of sets for a list, a collection and a dictionary.

Below, the operation selecting objects of design entities with a structure 'STR-5' is shown.

```
DesignEntity
  where: (versionDictionary includes: (Structure
    where: (identifier = 'STRU-5')))  
```

An implementation of this kind of operation for a given design data structure is realized through a declaration of a selection method 'where:' or declaration of its derivatives in the Entity/Element class. This method is then inherited through all classes of the data structure. The SMALLTALK realization of this method looks as follows:

```
where: aCondition
  objectList: attributeFullNameList
  objectList: := self allInstances.
  attributeFullNameList: := self allInstVarNames.
  ... 
```

A selection method is inherited through all classes of a data structure. One can use a number of conditions as criteria for selection. Subsequent conditions may be separated with the keywords 'and:' or 'or:'. A selection of 'FLIP' objects, being elements of a generic type 'FLIP_FLOP' serves as a short example.

```
Component
  its: actualParameterList
  where: identifier = 'FLIP'
  and: genericTyp identifier = 'FLIP_FLOP'.
```

A realization of this operation is similar to the realization of a one-condition operation.

6 Application of the object-oriented data model in ODE

The environment proposed in the paper enables representation of digital circuits in all main representation domains. Additionally, following some VHDL styles ODE supports also mixed structural and functional descriptions. It has been implemented in a typical representative of object-oriented languages, namely in SMALLTALK. The ODE environment presented in Figure 8 consists of: a design management unit, a graphical representation unit, a port definition unit, and functional and structural design units. Additionally, the environment provides some support for floorplaning and layout design.

```
library
  DesignManager
    PartDesign
    ShapeDesign
    BehaviourDesign
    StructuralDesign
    FloorplanDesign
    LayoutDesign
```

*Figure 8. A structure of the ODE environment.*

The management unit being a system's kernel enables opening and closing of a design library, opening of a graphical design window, introducing of design ports, opening structure and functional design windows, as well as storing of a current design in a library. The system allows for an arbitrary number of designs being "opened". Every design is represented through its design window of a management unit. This allows for opening different designs during a design process.

The class Window is a basic class (super class) of ODE. It defines attributes and methods enabling operation with graphics in design views (e.g., opening and closing of a window). Views may be either textual or graphical. In the Window class a global attribute MLibrary has been defined. This attribute points to the library (Figure 9), which groups other libraries and digital circuit designs. As a class attribute it is accessible in all design views, which inherit attributes of the class Window (classes: DesignWindow, ShapeWindow, BodyWindow, PureGeometryWindow). A contents of the MLibrary is briefly depicted below:

```
MLibrary:=
  <Library-1 (DesignEntity-1, ..., DesignEntity-n),
  ..., Library-m (DesignEntity-1, ..., DesignEntity-k)>
```

*Figure 9. A contents of the MLibrary.*

A declaration of features of design bodies is possible with design view editors of: a function, a structure, a floorplan, and a layout (classes BehaviourView, StructureView, FloorplanView and LayoutView). Every design may compose an arbitrary number of different bodies (Figure 10), which may also depend on each
other (e.g. a floorplan depend on a structure). A set of design bodies of a design circuit \( X \) we note as:

\[
I := \langle \text{Behavior-1, ..., Behavior-n, } \\
\quad \text{Structure-1, ..., Structure-m, } \\
\quad \text{Floorplan-1, ..., Floorplan-k, } \\
\quad \text{Layout-1, ..., Layout-l} \rangle
\]

Figure 10. Bodies of a design entity.

A graphical editor of design structures is one of the implemented tools. Another tool to be mentioned is a layout editor (class PureGeometryWindow). Using this editor, one may declare in ODE a list of geometrical objects, with technological layers being assigned.

7 Discussion of results

This section will discuss some technical advantages obtained as a result of application of the object-oriented data model in ODE.

1. The object-oriented development strategy supports a meet-in-the-middle design methodology [3]. The advantage of this methodology is that the final form of a design depends on the complexity of primitive cells, whereas the complexity of these cells may be arbitrary, i.e., from logical gates to functional blocks.

2. A structure of a digital system is defined through structural components—objects of the Component class. Instantiation of a structural component doesn’t introduce any major delays to an interactive design. This results from the fact, that the object of the structural component represents only the interface features of this sub-component. It points to the object representing its design, which is stored in a library. Figure 11 provides a diagram showing a relationship between an instantiation time of a decoder’s structure and a number of its gates for the above mentioned cases.

This diagram provides an evidence, that the instantiation time of a structural component depends relatively weak on a number of decoder’s gates. Further experiments on more complex components have confirmed this conclusion.

As mentioned above an instantiation time of a component depends on its interface. Due to that, and taking into account that interfaces of components differ only in a number of ports and a declaration of their generic parameters, we may anticipate a small instantiation time of the entity elements. This is illustrated in Figure 12 below, which comprises a diagram relating the instantiation time of a structural component of a decoder to a number of its ports.

Figure 11. An instantiation time of a decoder’s structure: a decoder as a design [1], a decoder as a component [2].

Figure 12. A diagram of the instantiation time of a decoder.

The diagram shows that even for a number of ports equal to 21 (four-bit address decoder) the instantiation time of a structural component doesn’t exceed 2n.

A designer can thus introduce new components representing “ready designs” as a part of a new design with a negligible instantiation time delay. From the above discussion we may conclude, that an introduction of an object-oriented features to a CAD environment in general results in reduction of time required for creation of a component object. This is of prime importance especially during an interactive design.

8 Summary

The ODE design environment employs the object-oriented data model for representation of a design data semantics. This has simplified a process of automatic data acquisition and analysis. Besides an object-oriented representation of design data, also CAD tools may be modelled as objects. This guarantees a unified access to CAD system resources.
The ODE environment data model is of a universal character. It enables design data representation for most design tools. The proposed data model exhibits also additional advantages by:

- comprising main representation types and abstraction levels of a digital system,
- integrating data of different description domains with tools of a CAD environment,
- including data for simulation and synthesis,
- simplifying a comprehension of complex designs.

Different tools either for synthesis or for verification from the ODE environment require some kind of performance evaluation. Design aspects to be evaluated, like a delay of a signal in a wire, or a silicon area taken by a design need a complete access to design data. This however demands objects being constantly present in a computer memory. If digital system components are declared as objects, then all required data are available through objects existing in a memory and created from classes of a data structure.

In general, founding of a data structure of the ODE object-oriented data model on VHDL and EDIF has simplified an interface between ODE environment tools and the tools based on existing standards.

References


