Visual polymorphism in schematic capture for VHDL-AMS

D.Parsons, Southampton Institute,
T. J. Kazmierski, University of Southampton

Abstract

This paper describes a new approach to schematic capture for VHDL-AMS that uses visual polymorphism to represent multiple underlying component models with a single visual invocation. The proposed idea provides a unified treatment of component connectivity at the interface point between the analogue and digital domains at the schematic level whereas the corresponding VHDL-AMS port definitions reflect the details of the connectivity context.

Introduction

Visual polymorphism in schematic capture for VHDL-AMS [1] is a new idea derived from the application of object technology to graphical entry systems. Object technology has been applied to a number of schematic packages, using a range of tools such as Smalltalk [2], C++ [3] and Object-Oriented Databases [4]. Polymorphism (the ability of different classes of object to respond differently to a single message) is a key component of object-oriented systems, though its documentation in the published research is frequently confined to the context of graphical user interface routines. In this paper, we use the term 'visual polymorphism' to describe how a single visual representation of a component may be used to invoke instances of different VHDL-AMS model descriptions, depending on its context within a mixed mode circuit.

Visual polymorphism

There are several coding techniques that can be described as exhibiting polymorphism. Cardelli and Wegner [6] classify four types of polymorphism: coercion, overloading, parametric polymorphism and inclusion polymorphism. Of these it is the latter form, defined by the object methods in a classification hierarchy, that is generally regarded as the key characteristic of a truly object-oriented system. However, what defines each of these as a type of polymorphism is the role of context in interpreting a symbol. In source code, the symbols to be interpreted are function names or operators, and the contexts upon which their behaviour depends are other associated code elements, be they parameters, data types or object classes. The concept of visual polymorphism, introduced in this paper, has much in common with these commonly recognised forms in that once again it is the role of context that renders a given symbol polymorphic. The symbols are not in this case code tokens, rather they are the visual representations of electronic circuit components and their context is connectivity relationships with other components.

VHDL-AMS syntax has much in common with class based programming in that models of components can be defined in terms of both their public interface and internal behaviour. Such encapsulation allows different implementations to be tested behind a single interface when instances of these models can be invoked as discrete objects [7]. There is also the opportunity to apply some aspects of object-oriented design using aggregation to provide a flexible layered architecture [8]. However, there is no true object-orientation in the language because it does not allow either inheritance or polymorphism. This is a particular problem when modelling mixed mode circuits because different models of digital components must be explicitly invoked to match the digital or analogue natures of the other components they connect to. In an object-oriented system, object behaviours can be overloaded to respond differently in different contexts in a process invisible to the programmer invoking the object. In VHDL-AMS, the process must be explicit, with the responsibility on the programmer to identify which model must be used in a given context.

VHDL-AMS schematic capture with visual polymorphism

In the absence of VHDL-AMS code mechanisms to invisibly invoke different model definitions depending on context, we can pass this responsibility to the schematic capture package. Thus visual polymorphism is the interface between a polymorphic symbol on screen and non-polymorphic invocations of component models in the generated code.

VHDL-AMS is a superset of VHDL'93 which provides a mechanism for describing the behaviour of both continuous and discrete components. The visual polymorphism approach in our schematic capture system provides a unified treatment of component connectivity at the interface point between the discrete and continuous worlds whereas the corresponding VHDL-AMS port definitions reflect the nature of signals depending on their connection context. Digital components have input and output signals that by default are digital signals, but change their nature when connected to analogue components. For example, the phase detector in figure 1 comprises nine Nand gates that are all by nature digital, but two of them have an analogue input. This is a screen dump from the VHDL-AMS schematic capture system. VHDL-AMS requires that these two components be modelled differently from the other 7 that have only digital inputs. The different models might be given these component declaration statements in the phase detector architecture:

First, a fully digital model of a Nand gate

```
COMPONENT NandD -- Nand gate with digital inputs
  GENERIC
  (
     delay: Time := 1 ns;
     width: positive := 2     -- number of inputs
  );
  PORT
  (
     SIGNAL inp:  IN bit_vector(width DOWNTO 1);  -- input signals
     SIGNAL outp: OUT bit                         -- output signal
  );
END COMPONENT NandD;
```

In contrast, a different model with a different local port clause is needed whenever some inputs may be analogue, such as this 'NandD_AD' gate.

```
COMPONENT NandD_AD   -- Nand gate with digital and analogue inputs
  GENERIC
  (
     delay: Time := 1 ns;
     AnalogWidth: natural := 1      -- number of analogue inputs
     DigitalWidth: natural:= 1      -- number of digital inputs
  );
  PORT
  (  -- analogue inputs:
     TERMINAL AnalogInp: electrical_vector(AnalogWidth DOWNTO 1);
     -- digital inputs:
     SIGNAL DigitalInp: IN bit_vector(DigitalWidth DOWNTO 1);
     -- output signal:
     SIGNAL outp: OUT bit
  );
END COMPONENT NandD_AD;
```

This difference in models demands that a writer of VHDL-AMS code must select the necessary model from a range of possibilities when building a circuit, even though the components represented are in fact the same. Writing the architecture for the phase detector involves deciding which of the gates are 'NandD' models and which are 'NandD_AD'.

This additional task for the circuit designer can be regarded as both tedious and error prone therefore a software solution would be of benefit. The selection of appropriate models would then be handled by the software. This is implemented within the schematic capture system by identifying the natures of the waveforms received and generated by the components. A single component object exhibits a form of polymorphism by generating different VHDL-AMS code depending on the type of model that needs to be invoked.

Like all types of polymorphism, different behaviour of a single type of object is driven by context. In this application, the different contexts are digital and analogue domains. Any nodes that connect components in the schematic capture package can be either digital signal ports or analogue terminals of the electrical nature depending whether that node carries an analogue or a digital signal. This difference is then reflected in the architecture definitions of the components to which the nodes are connected, and this is subject to change as the circuit design is edited. A node that connects only to digital components is a digital signal, but if it connects to at least one analogue component then the entire node becomes an analogue terminal of the electrical nature. Where such a node connects with a digital component, then the connection to that component will require the invocation of a different VHDL component definition. All components have the ability to detect the type of the nodes to which they connect, and invoke appropriate models when requested to generate VHDL-AMS code. The first element of the generated source code is the library context clause list followed by entity declaration. Then the architecture is defined, in which the necessary models are selected from a component library and included in the source. Once the models have been identified and loaded from the library file, the architecture body with component instantiations is described.

The following source code is generated from the phase detector in figure 1 by the schematic capture system:

```
LIBRARY IEEE, Disciplines, Mixed;
USE IEEE.math_real.ALL;
USE Disciplines.electrical_system.ALL;
USE Mixed.polymorphic_package.ALL;

ENTITY PhaseDetector IS
     PORT
     (
        TERMINAL Node9: electrical;
        TERMINAL Node11: electrical;
        SIGNAL Node8: bit;
        SIGNAL Node10: bit;
     )
END ENTITY PhaseDetector;

ARCHITECTURE Structure OF PhaseDetector IS
SIGNAL Node1, Node2, Node3, Node4, Node5, Node6, Node7 : bit;
BEGIN
    Nand1: NandD GENERIC MAP (d_width => 2)
           PORT MAP (bit_vector(Node1&Node2), Node3);
    Nand2: NandD GENERIC MAP (d_width => 2)
           PORT MAP (bit_vector(Node3&Node4), Node2);
    Nand3: NandD GENERIC MAP (d_width => 2)
           PORT MAP (bit_vector(Node5&Node6), Node7);
    Nand4: NandD GENERIC MAP (d_width => 2)
           PORT MAP (bit_vector(Node4&Node7), Node5);
    Nand5: NandD GENERIC MAP (d_width => 3)
           PORT MAP (bit_vector(Node3&Node&Node4), Node8);
    Nand6: NandD_AD GENERIC MAP (a_width => 1, d_width => 1)
           PORT MAP (electrical_vector(Node9),
                                  bit_vector(Node8), Node1);
    Nand7: NandD GENERIC MAP (d_width => 3)
           PORT MAP (bit_vector(Node6&Node7&Node4), Node10);
    Nand8: NandD_AD GENERIC MAP (a_width => 1, d_width => 1)
           PORT MAP (electrical_vector(Node11),
                                  bit_vector(Node10), Node6);
    Nand9: NandD GENERIC MAP (d_width => 4)
           PORT MAP (bit_vector(Node6&Node1&Node7&Node3), Node4);
END ARCHITECTURE PhaseDetector;
```

As well as identifying the names and types of the connecting nodes, the schematic capture system can also translate the physical sequence of input nodes into the logical sequence required by the component models, so that a NanD_AD gate with both digital and analogue inputs has to have its analogue inputs provided as parameters before its digital inputs. For example, gates 6 and 8 both have one digital and one analogue input but not in the same physical order. The schematic capture has automatically put node 1 as the first parameter to gate 6 to match the underlying model description. This process is invisible to the designer who can physically connect the inputs in any sequence without causing erroneous invocations of the underlying component model.

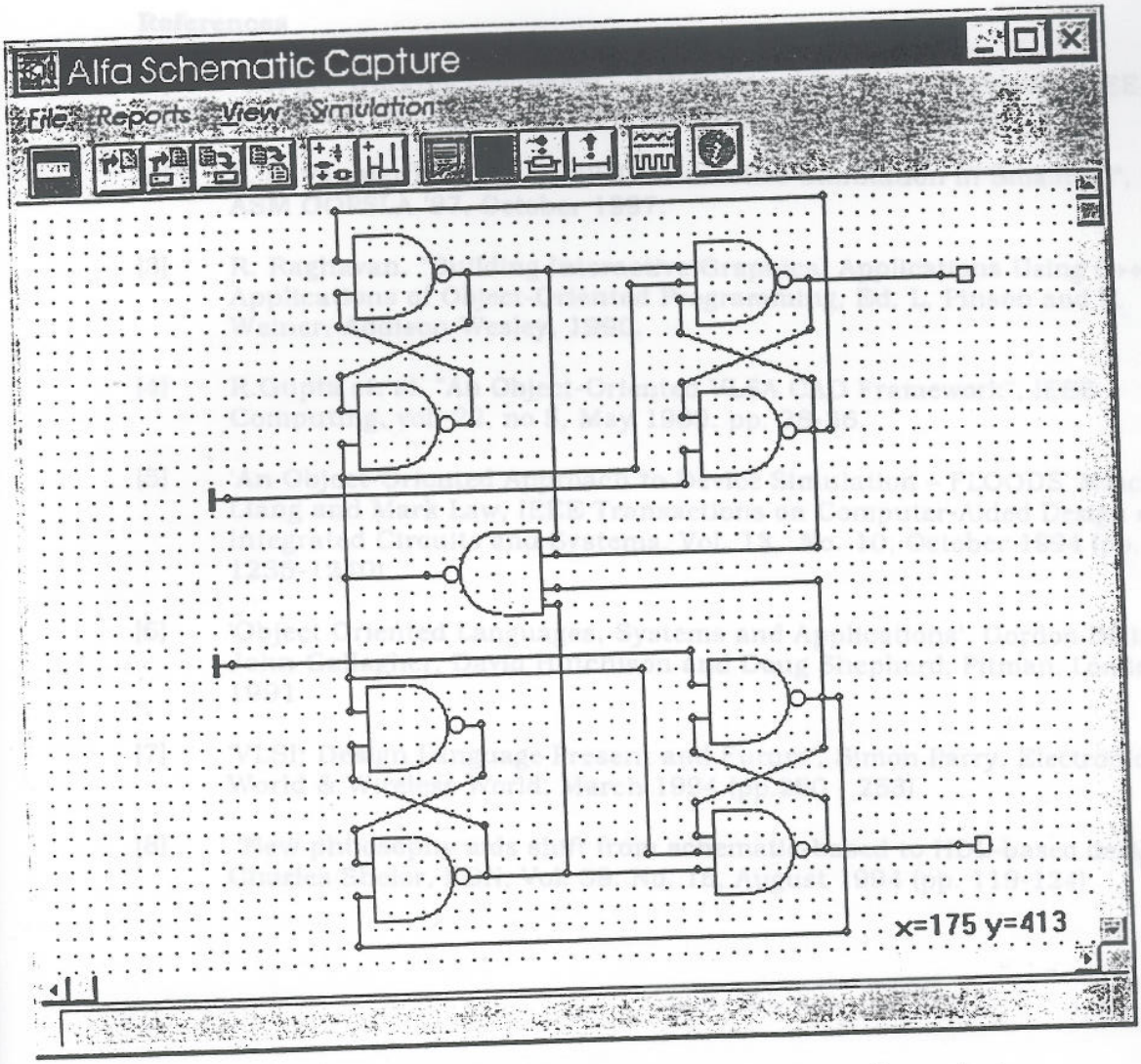

Figure 1. Phase detector schematic in the VHDL-AMS schematic capture system.

Conclusion

The schematic capture system described here uses visual polymorphism to render the design of mixed mode circuits transparent to the end user, in that only one visual model is placed on the screen. We define the concept of 'visual polymorphism' as a single visual representation of a component, which invokes instances of different VHDL-AMS configurations, depending on its connectivity context. Visual polymorphism is the interface between a polymorphic symbol on screen and non-polymorphic VHDL-AMS code describing component models. It is especially useful in a mixed-signal schematic environment, in which the natures of analogue and digital ports are implicitly derived from the visual representation. Since VHDL-AMS has no code mechanisms to invisibly invoke polymorphic descriptions, we pass this responsibility to the schematic capture package.

References

[1] "1076.1 Working Document - Definition of Analog Extensions to IEEE Standard VHDL", IEEE 1076.1 Working Group, May 1997.

[2] P. S. van der Meulen, "INSIST: Interactive Simulation in Smalltalk", Proc. ASM OOPSLA '87, October 1997.

[3] R. Raghavan, "Building Interactive Graphical Applications Using C++", in Applications of Object-Oriented Programming, Ed. L. Pinson and R. Weiner, Addison-Wesley, 1990.

[4] R.Gupta et. al. "An Object-Oriented VLSA CAD Framework", IEEE Computing, vol. 22, no 5, May 1989. pp. 28-36.

[5] 'An Object-Oriented Approach to Device Simulation – FLOODS' Minchang Liang and Mark Law, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 13, No. 10, October 1994 (pp. 1235-1240)

[6] 'Object Oriented Languages, Systems and Applications', Gordon Blair, John Gallagher, David Hutchison and Doug Shepherd, Pitman, London 1991

[7] 'VLSI: Design Language Present and Future', Simon Parry, Electronics World & Wireless World, March 1994 (pp.250 - 253)

[8] 'New philosophy aids shift from schematic-based to HDL-based design', Charles Shelor, EDN, Vol. 39, No. 16, August 1994 (pp. 119-124)