XML-Based Hierarchical Description of 3D Systems and SIP

Susann Wolf, Andy Heinig, Uwe Knöchel
Fraunhofer Institute for Integrated Circuits, Design Automation Division (IIS/EAS), Dresden, Germany
(Susann.Wolf| Andy.Heinig| Uwe.Knoechel)@cas.iis.fraunhofer.de

1 Motivation

3D integration of electronic systems offers new opportunities for designing smart products. One advantage is the dense integration of electronic systems consisting of inhomogeneous sub-modules (e.g., analog, digital, RF, sensors) which cannot be integrated in a single silicon technology. Another opportunity is the delay reduction by shorter interconnect length between multi-processors and memory in a stack which results in a higher performance. These systems will contain at least two dies interconnected by some hundred TSVs or other connectors. Their efficiency (costs, time response and thermal characteristic) can be improved significantly if the opportunities and risks of stacking are addressed at all design levels. The layout of a stack influences manufacturing cost as well as performance, which strongly depends on parasitic physical effects such as heat propagation and capacitive coupling.

The high complexity of 3D systems sets strong performance demands on the design flow. Much more than in SoC design, the layout information of a 3D stack must be considered at early design stages with an appropriate abstraction level. Design space exploration and floorplanning require circuit data in terms of netlists as well as additionally information from other design domains like circuit layout, assembly and packaging technology.

Figure 1: Application of the XML-based description language in 3D circuit design.
Current design tools for integrated circuits store layout information in a set of two-dimensional layer descriptions. The GDSII-format is commonly used for data transfer to semiconductor foundries. But if data for the 3D geometries in a stack are needed, e.g., for parasitic characterization of bond wires, no standardized data formats are available. Instead several proprietary formats exist, dedicated to a single design task, domain, or tool suite. This results in difficulties for exchanging 3D geometry data between different electronic design automation (EDA) tools. The design of 3D stacks demands the tight interaction between EDA tools and multi-physic tools for analyses of thermal, thermal-mechanical and electromagnetic effects as shown in Figure 1.

To overcome this bottleneck a common description format for storage and exchange of hierarchical structured 3D geometry data is proposed in this work. The major aim is to support data exchange at different design stages and with the appropriate level of detail. E.g., for thermal analyses a die is approximated by a cuboid of mixed material while electromagnetic analysis may require an accurate model of its metal layers. Since optimization of designs involves several iterations, the efficiency of data exchange is essential. Requirements to the proposed description language are:

- Compact storage of 3D geometry (less overhead)
- Support of hierarchical description, to deal with sub-modules in a design
- Support of parameterized and fixed elements at every hierarchical level
- Support of different abstraction levels to reduce simulation complexity
- Possibility for future extensions (e.g., adding descriptors for electrical parameters)

2 Description Formats for Geometric Data

To achieve a compact storage of complex designs, the geometric data is modeled by means of constructive solid geometry (CSG) [1], i.e., Boolean operations, which are applied to solid objects. In this approach the data are stored in a tree. Its root represents the entire system. The nodes represent subsystems such as ICs on a board as shown in Module A2 of Figure 1. Each of these nodes can be divided into sub-nodes. Figure 1 shows this for the IC which is partitioned into a package and a die with balls on a board.

We selected the mark-up language XML [2, 3] as basis for the description format because of its flexibility, extensibility and independence of computing platforms. XML is an open and widely-used standard well-supported by open source and commercial tools. It can represent CSG-based data well, because it stores data in a tree.

The CSG approach differs from approaches known from mechanical engineering, at which objects and surfaces are usually described as free formed surfaces, for example boundary representation (B-Rep, [1]). B-Rep represents surface or volume models by their boundary surfaces. These approaches are supported in the data formats IGES [4] and STEP (Standard for Exchange of Product Model Data ISO 10303, [5, 6]). In addition to IGES, STEP can represent data of the entire product lifecycle. The comprehensive STEP-model contains different parts such as material, surface data or geometry models. Further advantages of STEP are its modular standard definition, the implementability of subsets, and the convertibility between different information levels. Therefore, STEP starts to replace other exchange formats (e.g., IGES). Non-rectangular surfaces can be described well in these formats but this leads to large data size. In case of 3D integration most of the surfaces can be described with cuboids, spheres or cylinders. Therefore, CSG-based representation is very memory-efficient.

The Electronic Design Interchange Format (EDIF, [7]) can represent a complete design in one file which includes schematic, netlist and printed circuit board (PCB) or multi-chip modules (MCM). EDIF 4.0 was continued in the application protocol AP210 of STEP, but in general only a few tools support this version. EdaXML [8], an EDA-specific XML format, is based on the same information model as EDIF 4.0. XML-based specifications are a result of the IP-XACT [9] standardization activities. CMX (common model exchange) defines the structure of XML documents for description of hardware and software regarding IP (intellectual property) or device models. However, problems of 3D integration are not specifically considered there. EDA vendors support XML formats in some flows. E.g., the PCB tool Allegro by Cadence allows import and export of EdaXML data and import of Si2 EC PinPak XML [10]. However, these formats limit
oneself to the special field PCB application. Also Cadence can access the ANSYS HFSS tool which supports 3D electromagnetic field simulation, with a proprietary technology.

3 Transformation of XML to Other Languages

The capability of transforming XML data into other formats is the main advantage using this language for the data management in a design flow with different tools such as floorplanning, package design, and electric, electromagnetic and thermal simulations. Because of this, XML is already widely applied in the EDA environment.

A transformation into the XML format X3D (extensible 3D) [11] allows a visualization of 3D geometries with little effort. X3D is a data format for storage of modeled 3D sequences, which is flexible, extensible, and modular.

A further advantage is data transformation (e.g., geometry, material, heat conductivity or hierarchy) into a data exchange format supported by geometry based simulation tools such as ANSYS or CST. Many different CAD interchange formats exist. Based on a literature survey we decided for the neutral and compact exchange format STEP from mechanical engineering provided modeling with CSG and B-Rep, modular principle or level concept. Although STEP allows modeling with CSG, tests revealed that CSG feature is not supported by most tools used in 3D circuit design.

4 Basic Concept

In this section, we describe the hierarchy and the coordinate system as the underlying concepts of the proposed description language.

4.1 Hierarchy

The complexity of the 3D systems requires a hierarchical description. On the lowest level we define primitives such as Cuboid, Sphere or Cylinder, see section 5.1. From these we derive basic elements such as Balls or TSVs, by definition of further parameters like material properties in addition to the geometry of primitives (section 5.2). They can be arranged to hierarchical elements that may consist of Balls, TSVs or hierarchical groups of such. This allows the reuse of previously defined structures at different positions in a system and at different times in the chip design. For example, a redundant TSV is composed of two single TSVs.

The composition of solid objects from basic elements allows an efficient and well-structured data storage. A complex 3D model is generated on basis of three-dimensional primitives by hierarchical execution of basic operations using the CSG method. The primitives are solids, i.e., they possess a closed surface. Furthermore, the developed hierarchical XML-based structure can be visualized by means of a tree (see Figure 2). The root of the tree represents the 3D system of arbitrary complexity as a final result. The basic elements are at
the lowest level. The element at the level above arises from the application of one of the operations defined in section 5.3 on two basic elements. This principle is continued step by step to describe the entire system.

4.2 Coordinate System

The coordinate system has to support the hierarchical composition of systems from basic elements and subsystems that can be placed repeatedly. Grouped units result from operations that are applied to 3D basic elements.

Each element, group or system has its individual coordinate system with a specified point of origin as reference point which is located at (0,0,0) by definition as shown in Figure 3. All elements of a group are placed with relative coordinates referring to the point of origin of the group. The statement <position> specifies where the point of origin of the coordinate system of the element is placed in the coordinate system of the group. For primitives the position of the point of origin is defined by the geometric form. For Spheres, the point of origin is located in the center, for Cylinders in the lower center point and for Cuboids in the front lower left corner. The 3D basic elements derive this information by the <include> statement from the primitives. The example in section 5.3.1.1 illustrates the application of the coordinate system.

Relative referencing using the distance between two objects is also allowed. Furthermore, translational displacement and rotation of objects are useful and necessary instruments. The translation could be realized by modification (addition/subtraction with the translation shift) of the appropriate coordinates of the elements considering relative data. To facilitate rotation, the point of origin of primitives could be located at the rotation axis. Additionally, angles of rotation are still to be specified. We aim to describe rotations and translations by means of transformation matrices. Therefore, the position data $x$, $y$, $z$ is calculated by application of the transformation matrix and afterwards the respective operation is executed.

4.3 Support of Different Abstraction Levels

The language shall furthermore support the description of different abstraction levels of a component or subsystem. For example, a rectangular TSV with its truncations due to manufacturing can be modeled on a more abstract level using a Cuboid. The selection of the appropriate model depends on the design task. For example, the characterization of electromagnetic parasitic effects demands an accurate description of geometry, whereas the simplified model is sufficient for thermal analyses.

5 XML-Based Description Language

In this section, we introduce the syntax of the proposed description language. It consists of primitives, from which basic elements of 3D systems (e.g., Balls and TSVs) are derived. Mathematical functions are provided to built-up modules, subsystems, and systems in a hierarchical way.

5.1 Primitives

High data volume accrues during the description of typical 3D systems. Therefore, primitives must efficiently represent these systems. One way is to define many different primitives which have only few parameters. However, since a large number of primitives reduce the clarity of description, the quantity of primitives should not be too large. The resulting task is to find a good balance between the number of basic elements and the number of parameters of each element. With regard to the description of 3D systems we define the parameterized primitives Sphere, Cylinder, and Cuboid as follows:

```xml
<element name="Sphere">
  <interface>
    <field name="Diameter" type="double" />
  </interface>
  <diameter d="Diameter" />
  <!-- point of origin located in center of Sphere -->
</element>
```
5.2 Basic Elements for the Description of 3D Systems

Basic elements can be composed of primitives, which are imported by using the keyword <include>. Therein, the attributes of the primitives can be redefined optionally. In the non-parameterized basic element TSV 20um the size of the primitive Cylinder is set to a diameter of 20 µm and a height of 100 µm. Additionally, <material> describes the material, for example copper or tungsten. The following listing shows important basic elements of 3D systems: SiliconDie, TSV, TSV 20um, and Ball, whereby SiliconDie, TSV, and Ball are parameterized elements. Other basic elements such as SLID or bump can be defined in the same way.

Geometrical characteristics are specified for each primitive. The statement <interface> specifies name and type of the parameter of the element. For example, DiameterHeight declares diameter and height as parameters of a cylinder.
The keyword `<statement>` expresses a general descriptor (string) and `<scale units>` specifies the scale units for the dimension-less given parameters. So far scale units are length specifications. For future extensions concerning modeling of thermal and electrical characteristics, it should be also possible to distinguish between several scale units. In addition, we also specify an `<interface>` which can be used to pass values of an element definition into an element definition of higher hierarchical level. The `<field>` keyword defines the parameter, which is set by `<fieldValue>` in the `<include>` statement. This is subsumed by the operation `Insert` combined with the positioning of the included element and its material specification.

### 5.3 Operations for the Composition of Systems

Applying operations allows the composition of complex systems from several basic elements. We distinguish between primitive and composed operations.

#### 5.3.1 Primitive Operations

##### 5.3.1.1 Union

Using the union operation (Unite, $\cup$) two or more elements can be connected to a new element (e.g., shape of first element $\cup$ shape of second element). Therefore, the new element represents a composition.

Unite connects basic or composed elements, which are imported using `<include>`. The composition has a coordinate system with the point of origin at (0,0,0). `<position>` specifies the location of the included elements with coordinates $x$, $y$, $z$ relative to the point of origin of the composition. As Unite represents a union from geometric point of view, the material data of the elements remains unchanged. The following example XML code describes the union of a die and two balls.
Figure 3 depicts the resulting composition and illustrates the handling of coordinates. The left-hand side shows the elements SiliconDie and the two Balls before union. Each of them has its individual coordinate system. The point of origin of the SiliconDie is defined as the front lower left corner of the element because it is derived from the primitive Cuboid. The center point of the included sphere is the point of origin of the balls. The function Unite builds the composition DieAndTwoBalls with the point of origin at (0,0,0). First the SiliconDie is placed at position (0,0,0). That means that the origin of the element is placed at the origin of the composition. The balls are placed with different coordinates. For example, the height $z$ is computed by the height of SiliconDie plus half diameter of Ball. The $<$position$>$ statement places the point of origin of the Ball (center point) to the desired positions. As we added half the diameter of the Ball to the $z$-coordinate, its surface touches the SiliconDie.

![Diagram of operands of Unite with their individual points of origin](image1)

![Diagram of composition DieAndTwoBalls with point of origin](image2)

**Figure 3:** Union of a die and two balls (not scaled).

The resulting composition can be used to built-up more complex compositions in a hierarchical way. The example below shows the union of the previous composition with an additional die. Again the $<$position$>$ statement determines where the points of origins of the elements are placed in the coordinate system of the resulting composition. In the example a second die is stacked on top of the DieAndTwoBalls composition.

```xml
<element name="DieAndTwoBallsAndDie">
  <interface>
  </interface>
  <operation name="Unite">
    <include name="DieAndTwoBalls">
      <position x="0.0" y="0.0" z="0.0" />
    </include>
    <include name="SiliconDie">
      <fieldValue name="LengthWidthHeight" value="5000.0 3000.0 100.0" />
      <position x="0.0" y="0.0" z="150.0" />
    </include>
  </operation>
</element>
```
5.3.1.2 Difference

Analog to the union operation we introduce a difference operation ($Cut, \setminus$). It cuts-out the shape of the second operand from the first shape. The result is an empty volume in the first operand. The resulting element (shape of the first element \setminus shape of the second element) is named as DieMinusTSV in the following example, which is depicted in the second row of Figure 4.

```
<element name="DieMinusTSV">
  <interface/>
  <operation name="Cut">
    <include name="SiliconDie">
      <fieldValue name="LengthWidthHeight" value="5000.0 3000.0 100.0"/>
      <position x="0.0" y="0.0" z="0.0"/>
    </include>
    <include name="TSV">
      <fieldValue name="DiameterHeight" value="20.0 100.0"/>
      <position x="1200.0" y="0.0" z="0.0"/>
    </include>
  </operation>
</element>
```

5.3.2 Composed Operation

Composed operations increase the description efficiency for complex systems. Currently the XML supports the operation $Cut_{\setminus}Unite$, which represents a difference operation followed by a union operation.

A common application is TSV modeling as visualized in Figure 4. At first, the shape of a TSV is cut out from a SiliconDie, which leaves an empty area. Afterwards the resulting element is unified again with the TSV shape. The materials of the elements remain during Unite. As result the TSV is inserted at the defined die position. In this operation the second and third operand have the same position. As the processing of the active silicon is not modeled at this abstraction level, via-first and via-middle result in the same model. The following listing shows the operation in detail:

```
<element name="DieMinusAndTSV">
  <interface/>
  <operation name="Cut_Unite">
    <operation name="Unite">
      <operation name="Cut">
        <include name="SiliconDie">
          <fieldValue name="LengthWidthHeight" value="5000.0 3000.0 100.0"/>
          <position x="0.0" y="0.0" z="0.0"/>
        </include>
        <include name="TSV">
          <fieldValue name="DiameterHeight" value="20.0 100.0"/>
          <position x="1200.0" y="0.0" z="0.0"/>
        </include>
      </operation>
      <include name="TSV">
        <fieldValue name="DiameterHeight" value="20.0 100.0"/>
        <position x="1200.0" y="0.0" z="0.0"/>
      </include>
    </operation>
  </operation>
</element>
```

An abbreviated description of this function is used to increase the clarity of the code as shown below.
Alternatively, DieMinusAndTSV could be represented by application of Unite to the elements DieMinusTSV and TSV. But the combined operation Cut_Unite is more efficient, because it reduces the number of operations and elements in a system description. Similar to the union operation more than one TSV can be placed in a single call.

### 5.4 Example

Figure 5 depicts the example of a packaged die. It contains a board, four balls, one die and a package compound.

For the modeling of this example we use the primitives Sphere and Cuboid. From those we derive the basic elements SiliconDie, Ball, PackageCuboid, and Board. First, we generate the group DieAndFourBalls using Unite and the elements SiliconDie and four Balls. Afterward Unite composes DieAndFourBalls and Board. Again Unite is applied to cuboids consisting of compound material to build up the element Package. At last, union generates the resulting element DieAndFourBallsAndBoardAndPackage.
The following summary of the source code shows the element DieAndFourBalls and the desired composition DieAndFourBallsAndBoardAndPackage:

```
<?xml version="1.0"?>
<PackagedDieModel>

  <element name="DieAndFourBalls">
    <interface>
    </interface>

    <operation name="Unite">
      <include name="SiliconDie">
        <fieldValue name="LengthWidthHeight" value="300.0 200.0 50.0" />
        <position x="0.0" y="0.0" z="50.0" />
      </include>

      <include name="Ball">
        <fieldValue name="Diameter" value="50.0" />
        <position x="45.0" y="30.0" z="25.0" />
      </include>

      <include name="Ball">
        <fieldValue name="Diameter" value="50.0" />
        <position x="115.0" y="30.0" z="25.0" />
      </include>

      ... <!-- "fourth Ball" -->
    
      ... <-- "fourth Ball" -->
    
      ... <-- "fourth Ball" -->
      ...
    </operation>
  </element>

  ...

  <element name="DieAndFourBallsAndBoardAndPackage">
    <interface>
    </interface>

    <operation name="Unite">
      ...
    </operation>
  </element>

</PackagedDieModel>
```

The XML file size of a 3 die stack with about 100 balls and TSVs is with 10 kB very small, compared to the corresponding STEP file with 1500 kB. The hierarchy concept of the XML supports memory-efficient modeling of realistic 3D systems with up to 2000 balls and 10000 TSVs.

6 Examples for Language Application in 3D Circuit Design

The systematic and machine readable description of three-dimensional geometries facilitates the data exchange between design and simulation tools in a design flow.

One application is the identification of contact areas for computation of heat propagation. The transfer of dissipated power in form of heat is a big challenge in 3D design. Structures of TSVs and metal layers transport heat from internal layers of a stack to the outside, where heat sinks can be attached. An analysis of the 3D geometry allows the identification and prevention of so-called hot spots in the design, which can lead to defaults in operation or reduced life time of the system.
It is intended to add the property of thermal conductance to the basic elements. For computation of heat propagation it is necessary to recognize the surfaces which provide contacts between the individual elements of a system. The proposed data structure supports different algorithms for contact detection.

Together with a description of design rules, the structured and machine readable description of 3D systems allows the check of design rules for stacking and packaging. Compared to the classical 2D design, more and different design rules must be considered:

- Keeping of minimum distances in three dimensions
- Overlapping of parts
- Contact between parts
- Processing order

The manifold of technological possibilities of 3D assembly increases the quantity and complexity of the design rules. Therefore, automated design rule checkers that can handle 3D geometries will increase the design productivity significantly. As the definition of a design rule set is a difficult task, a standardized and tool independent data format is essential. It is possible to add language constructs for design rule description to the XML language.

7 Outlook

This paper presented a hierarchical description format for the design of three-dimensional integrated electronic systems, which is based on the XML language. It will facilitate the data exchange between physical design and EDA tools as well as within EDA tools. The work outlined the general concept, which will be further developed and adjusted to requirements arising from practical applications. Our vision is a standardized modeling language for components, subsystems, and 3D integrated systems supported by all tools that need geometry and material data. In the future, the integration of electrical equivalent models of basic elements will be an important extension of the language. The definition of XML statements for management of abstraction levels is an ongoing research task. We aim to contribute the proposed format to standardization activities.

8 References

[9] IP-XACT: http://dx.doi.org/10.1109/IEEESTD.2010.5417309

9 Acknowledgment

Parts of this work were carried out within the research project NEEDS that is funded by the German BMBF under reference 01M3090.