PA-APIs: Looking beyond power intent specification formats

Amit Srivastava
amit_srivastava@mentor.com
Awashesh Kumar
awashesh_kumar@mentor.com
Mentor Graphics

***Abstract*- Increasing complexity of electronic systems and their power management has resulted in sophisticated power intent formats. UPF, the IEEE standard to specify power intent, has underlying semantics that minimize the effort of specification by deferring more complex processing to tools. There is a widespread impact of power management information in the overall design flow requiring tools and users (both design and verification engineers) to be aware of the result of the application of power intent information coming from different sources. However, due to limited capability in UPF and an inconsistent approach of getting the power management information through UPF query commands, users are forced to adopt ad-hoc approaches to get this information. So, there is a need for a well-defined and complete abstract data model to represent the processed power intent that can be accessed via a simple interface. The paper provides the details of an abstract data model for Power Aware (PA Information model) designs and the interface (API) to access the information along with highlighting its utility for different purposes.**

I. INTRODUCTION

The constant need for power efficient electronic systems and shrinking process geometries demands sophisticated power management architectures to meet the power budget. The SoCs have become incredibly complex and contain a variety of IPs ranging from processors, memories, peripherals and analog sensors. Some of the IPs are pre-implemented and have power management architecture built into them, while others have power architecture captured in UPF format.

Power aware design involves partitioning a design into separate regions, or power domains, such that each domain can have its power supplies managed independently. At any given time, only a subset of the full system may be powered, and a given portion may operate at different voltage and consequent performance levels depending upon the operating mode of the system. The ability to control power consumption based on functional requirements of the system is typically referred as power management.

Power management is imposed on top of the normal functionality of the design in a manner that is transparent to the user. The power management infrastructure can be thought of as a layer of logic integrated with the design logic that controls power supplies for each separate power domain, mediates the interfaces between power domains to ensure that differences in power state do not interfere with functionality, and orchestrates the transitions between power states to ensure that system state information is preserved where required or reinitialized as needed.

While power management logic historically has been added to the design late in the flow, the increasing need for power aware verification has led to methods for specifying power intent, i.e., the intended power management infrastructure, much earlier. A power intent specification is essentially a virtual definition of the power management logic that will be added in the implementation process, provided in a form that can be implicitly integrated with an RTL description in order to verify the RTL specification together with the planned power management infrastructure.

II. UPF

IEEE Standard 1801 Unified Power Format (UPF) [1] enables the specification of power intent. Based on Tcl, UPF defines concepts and commands required to describe power management requirements for IP blocks, the architecture of the power management infrastructure for a complete system, and even the implementation of power distribution networks. It is used both for early verification of power intent and to drive implementation of the power management infrastructure.

The UPF enables use of Tcl's scripting capabilities to ease capturing of the power intent. This reduces the burden on users and automates redundant tasks by using scripting capabilities. The power intent can also be split across various UPF files and can be restructured as per IPs of the system. Users can reuse the power intent that was used to develop the IP and use it in the SoC environment, especially when they are dealing with soft IPs.

In UPF, users can create various abstract objects that contain information related to power management, e.g. Power Domains, Power States, etc. These objects get refined further with more information as and when they are

available. It also defines various concepts to reduce the verbosity of specification and automate the application of power intent. Concepts like precedence rules allow users to create a general rule for a majority of tasks and a specific override for the exceptional cases.

A. Dependence on HDL

The UPF commands and semantics are heavily dependent on HDL information for capturing the power intent. All UPF objects are created in a specific HDL scope. The rules are defined in such a way that they can be applied to specific design hierarchy. Users can create rules that can result in specific placement of isolation/level shifter cells based on connectivity present in the HDL.

III. Access To Power Aware Information

Due to the overall effect of power management on the user design, the power intent has become an integral part of the design intent and is required at every stage of the design flow. Access to power aware information is not just significant for tool developers but also to design and verification engineers. They need to be aware of the power management architecture and query it to fulfill their requirements. Some of the more specific requirements for accessing power aware information include:

- Develop utilities for design intent exploration.
    - Generate a customized report of power architecture information for specific exploration. E.g. List of power management cells inserted in the design.
    - Generate a list of retention cells in the design at RTL stage to assess whether a full retention or partial retention strategy is needed.
- Develop utilities that help create additional UPF definitions.
    - Understand the power management of an IP and then accordingly create the power intent of the SoC depending upon characteristics of the IP.
- Develop custom checking utilities that can be incorporated into a quality checking flow.
    - Check that an isolation clamp value matches the reset value.
- Create an environment to generate coverage monitors and assertions using standardized Tcl queries.

IV. Obstacles To PA Information Access

There are various challenges to accessing information related to power management. Due to the separation of power intent from HDL, the HDL information model doesn't contain details about power management and hence users need to depend on tools to get that information.

The complexity of UPF specification, the scripting nature of UPF, and its reliance on HDL knowledge makes it difficult to interpret the UPF files in isolation to extract the relevant information. This creates a requirement of some mechanism to get access to power aware information.

Moreover, in most practical scenarios, the information related to power intent is not just confined to UPF. This information may be present in the library cell definitions or HDL descriptions in the form of special attributes, which users are unwilling to duplicate in UPF. Thus it's important for a power aware information model to go beyond the scope of UPF to capture information coming from various other sources and provide a combined representation of power aware information.

The UPF standard defines a set of Tcl queries to access UPF information. These queries are mainly based on accessing the details of the UPF commands and properties on them. e.g. query_power_domain. Often, this kind of query is not very useful as it mainly presents details that are present in UPF and not the result of application of those commands on HDL. There is a query_upf command that provides some options to query the results of application of UPF. But it is incomplete and lacks various details that render it useless. It doesn't provide enough details about the underlying information structure that needs to be queried and hence doesn't produce the expected results. To address the limitations of UPF, different tools provide their own implementations of these queries or even proprietary access mechanism to query the power aware information. This causes problems with interoperability and consistency across tools.

All these challenges force users to rely on different tools to get access to power management information for their unique requirements. Sometimes, there is a different interpretation of UPF language leading to discrepancy in information. It is therefore very important to define the model which captures the information about power management. This model can then be used across tools to provide a consistent interface and behavior.

V. Information Model

An information model is a representation of concepts, relationships, constraints, rules and operations to specify data semantics for a chosen domain of discourse. The advantage of using an information model is that it can provide sharable, stable, and organized structure of information requirements for the domain context [2]. The information

model captures the structure of information and not how it is implemented. Tools are free to represent the actual implementation in any language and data structures. However, an information model needs to be captured in a format that can be shared and used for reference. There are various languages available that can formally capture the information model, e.g. Integrated Definition Language 1 Extended (IDEF1X), the EXPRESS language and the Unified Modeling Language (UML). In this paper, we have used UML to capture the information model for power management.

A. UML

UML is a standard modeling language for specifying, visualizing, constructing, and documenting the information model [2]. It consists of set of diagrams like Class diagrams, Object diagrams, component diagrams, etc. In this paper, we are mainly concerned about the information model structure that can be represented as Class diagrams.

A UML Class diagram describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects [3]. It can easily represent the inheritance relationships and composition relationships along with the details of cardinality.

Similarly, an object diagram in the Unified Modeling Language (UML), is a diagram that shows a complete or partial view of the structure of a modeled system at a specific time. We have used an object diagram to represent various examples in the paper [4].

The following figure (Fig. 1) represents the various kinds of relationships that can be captured in UML class and object diagrams.

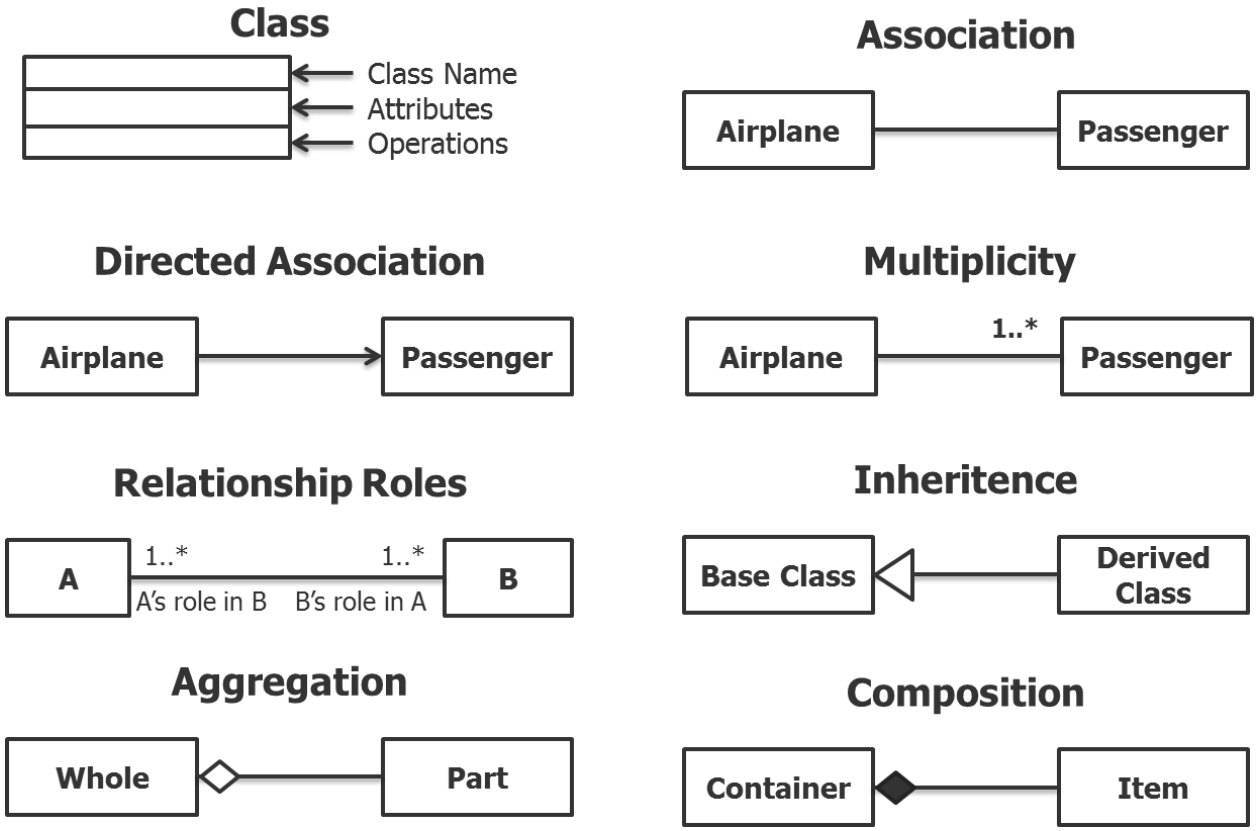

Fig 1: UML notations

VI. POWER AWARE INFORMATION MODEL

Power Aware Information Model (PA-IM) is a model that captures the information related to power management for a given design. This consists of set of objects containing information and various relationships between those objects. The PA-IM will form the underlying structure which will be accessible to the external world via Application Programmable Interface (PA-API).

The PA-IM will contain the result of the application of power intent over the user design. Hence, it represents objects which are created to capture power related information coming from UPF as well as a necessary subset of the user's HDL design, which is needed for the power management. Along with that it also maintains the

appropriate relationships between those objects, e.g. which instances in the user design belong to which power domains in UPF.

This well-defined structure, containing complete information about power management, will allow users to access information which was not possible from existing mechanisms. Such as,

- Which specific logic in the design is powered by which supplies? How does it related to UPF?
- What are the contents of the power domain and which instances does it power?
- What are primary supplies of the domain?
- What are isolation strategies in the domain?
- What cells are inserted in the design by the isolation strategies of this domain?
- What is the connectivity of an isolation cell, i.e. what are hiconns of isolation cell connected to?
- And many more similar questions …

The Fig. 2 captures the concept and construction of PA_IM and how it is accessed via PA-API.

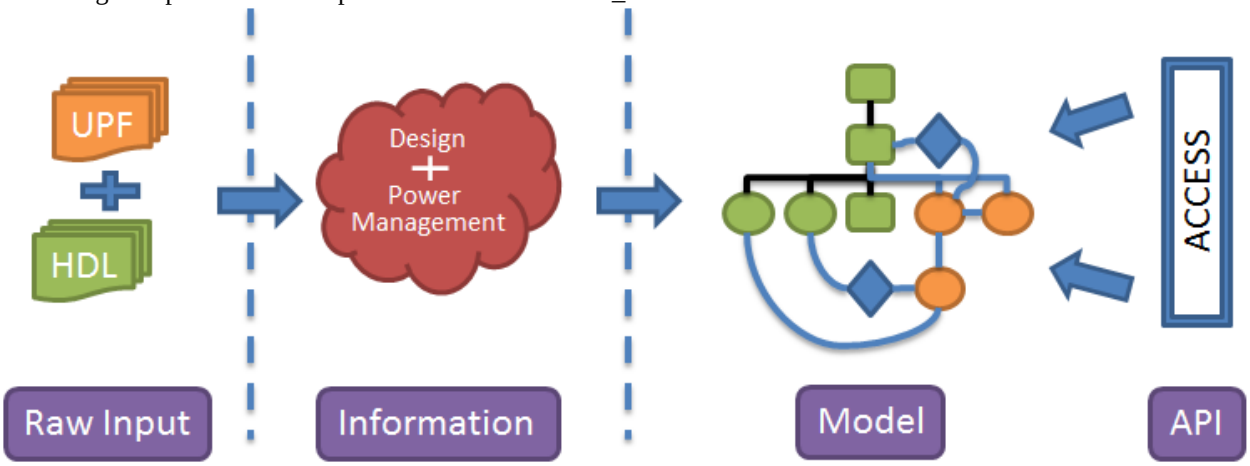

Fig. 2: Information Model

A. *Basic components of Power Aware Information Model*

The information model defines various classes that represent information about the power management. For a given design, various instances of these classes form the content of the information model. These instances are termed as Objects. The Objects contain information in the form of attributes.

Objects: These are the primary holders of PA Information. They are classified into three broad groups:

- UPF Objects
- HDL Objects
- Relationship Objects

Attributes: They represent the piece of information present on the objects with a predefined attribute name. They can be of following types:

- Basic types: String, Integer, Boolean, Float, Enumerated
- Complex: Handles to other objects or a List of handles to other objects

PA-Handle: A reference to a particular object in PA-IM is called a PA-Handle. The PA-Handle will be used for querying information from the information model via APIs.

VII. Power Aware Information Model Objects

PA-IM objects are classified in three major groups:

A. *UPF Objects*

The UPF objects represent a group of objects that are created by UPF for specifying power intent. They represent abstract objects that contain properties necessary to model the power management information, e.g. power domain, strategies, supply network, power states, etc. These objects have a direct mapping to objects created in UPF and store information directly derived from UPF. The objects also store the UPF source information in the form of attributes to allow a direct correlation.

UPF Version Compatibility

It is ensured that PA-IM is compatible with all versions of UPF. Most of the differences in UPF are related to how the power intent is captured and in enabling tool automation to build the power management architecture. Since, PA-IM captures the result of the application of power intent, which is more or less same after all the UPF commands are applied, therefore the objects in the PA-IM can represent any UPF version.

The attributes present on the UPF Objects are comprised of the superset of all the information and will only be populated when there is a UPF command that specifies them. This allows the PA-IM to be agnostic of the version of UPF used during the processing. As a result, these objects do not maintain any information about the version of UPF.

Non UPF Information

Since UPF Objects are designed to capture the result of power intent, it can also be used to capture power management information specified in some other formats like CPF. The different formats have similar concepts of capturing the power intent, although the actual processing may be slightly different but conceptually they will create the same power management architecture over the design. For example, a PA-IM object to represent a power domain represents the same low power idea regardless of the source format (UPF or CPF). It can also be used to capture power aware information present in library formats like liberty and even in HDL.

List of UPF Objects

1. Power Domain
2. Retention Strategy
3. Isolation Strategy
4. Level Shifter Strategy
5. Repeater Strategy
6. Supply Set
7. Supply Net
8. Supply Port
9. Logic Net
10. Logic Port
11. Power Switch
12. Ack Port
13. Power Switch State
14. Power State
15. Supply Port State
16. PST
17. PST State

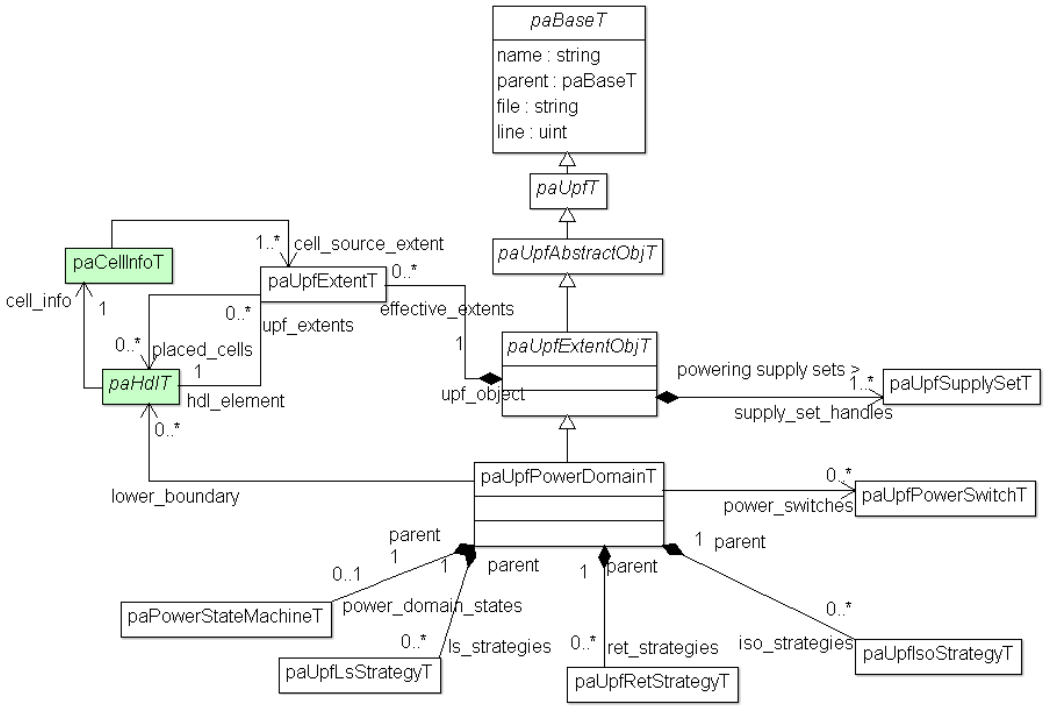

Fig 3: UML class diagram for Power Domain its components and associated classes

B. HDL Objects

The HDL Objects represent the information coming from HDL design. It captures abstracted HDL information that is independent of the language in which the design is written. Only skeleton information like name, size, structure, etc. is captured.

The HDL Objects present in the PA-IM are only a subset of actual design information. Since UPF relies on HDL information for representing the low power intent, only the objects those are necessary to capture the power aware information is present as HDL Objects in the PA-IM. The significance of these objects is to just provide a placeholder and starting point for PA information. They are not supposed to contain complete HDL information. If the user wants to access specific details pertaining to HDL, they need to use the respective HDL information models to access the information. The PA-IM will provide the full hierarchical pathname which is consistent with the HDL design. This pathname can then be used to query HDL specific information.

Representing Signal of Complex Objects

The PA-IM represents signals of complex types, like record, structure, and arrays, as a multi-bit kind of object. The signals of any complex type get transformed into a normalized vector of bits in the information model. This helps in providing a consistent and simple representation across all HDLs. The normalization is done based on a well-defined algorithm. Along with that, there is a dedicated API to return an un-normalized name for the multi-bit object so that user can draw a better correlation with the original RTL.

List of HDL Objects

Following, is the list of HDL Objects present in the information model:

1. HDL Scope: represents the instances present in the user design.
2. Ports
    a. HDL Scalar Port: represents a port in the HDL that requires a single bit representation.
    b. HDL Multibit Port: represents a port in the HDL that requires a multi-bit representation.
3. Nets
    a. HDL Scalar Net: represents a signal/net in the HDL that requires a single bit representation.
    b. HDL Multibit Net: represents a signal/net in the HDL that requires a multi-bit representation.
4. MultibitSlice: represents a slice of a multi-bit object that can either be a multi-bit port or net.

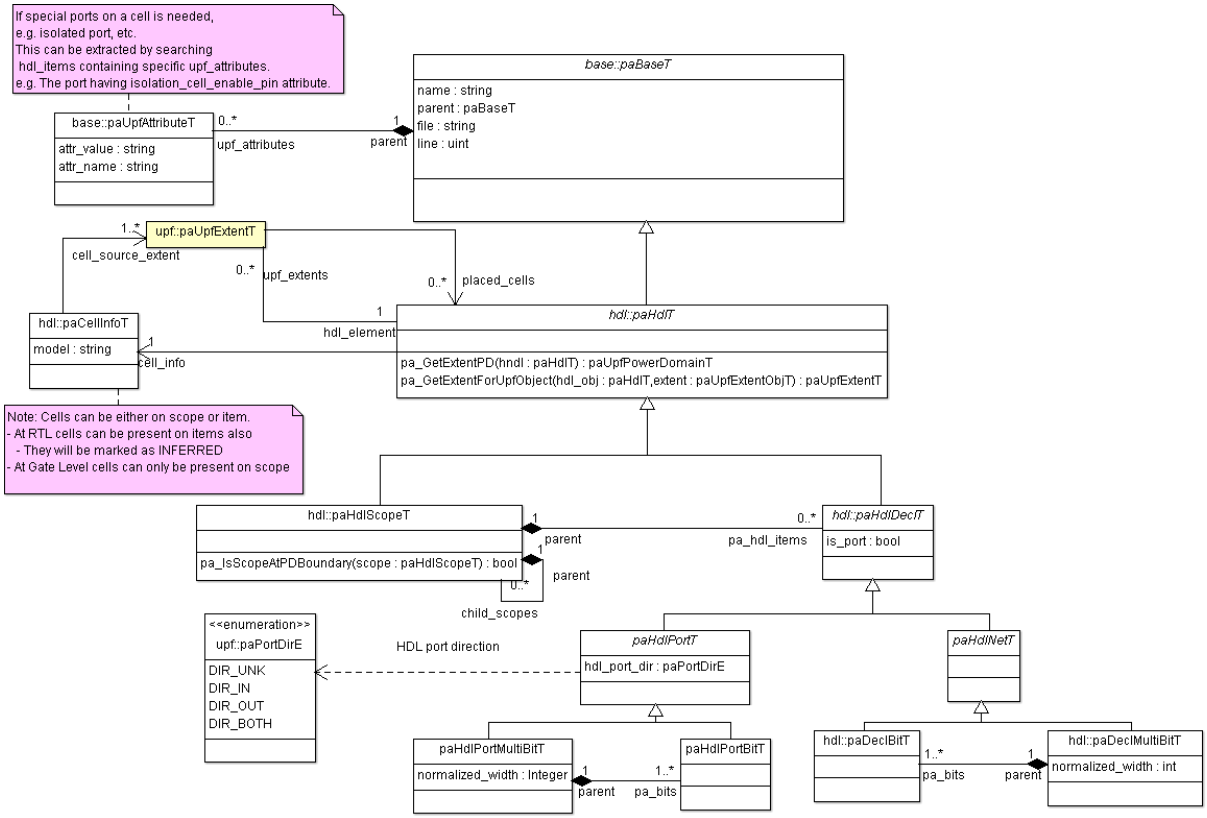

Fig 4: UML Class diagram representing classes for HDL Objects

C. Relationship Objects

The Relationship Objects are special objects created to capture various relationships between UPF Objects and HDL Objects. They serve as intermediate objects to provide bidirectional relationships and bridge the gap between abstract UPF Objects and real HDL Objects.

List of Relationship Objects

1. UPF Extent
2. Control sensitivity
3. Cell Information
4. Expression

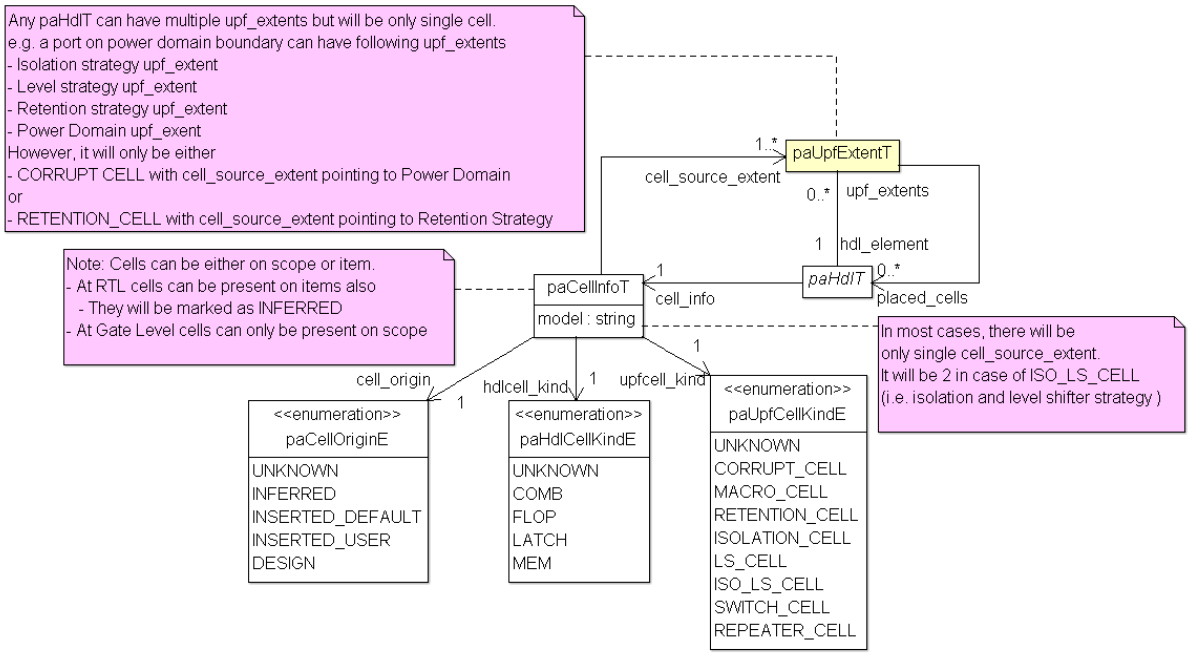

Fig 5: UML Class diagram representing Cell Information

VIII. PA-API

The Power Aware Application Programmable Interface (PA-API) is the interface that is used to access the contents of the PA-IM. Depending upon the requirements, the access can be required in various environments, such as C interface, Tcl interface, or an HDL interface like SV and VHDL. The interface in different environments will be similar in structure with minor adaptions for syntactical purpose.

The PA-API will provide a read-only access to the information model. They will be available after the UPF processing is done and power intent has been applied on the design. Because the information model is created from the application of UPF, the write access APIs to change the content are not required and hence are not present in the information model.

However, there is a need to modify certain dynamic attributes on the object during simulation, like current values or supply network objects. This is mainly needed for creating more advanced testbenches for manipulating the power management. The current work does not focus on the dynamic attributes. This will be done in the future.

In this paper we present two different interfaces for PA-IM in C and Tcl languages. Interfaces in other languages can be easily extended based on these APIs.

A. C APIs for PA-IM

The list of various functions that comprises the C interface of the information model is shown in Table 1.

C PA-API		
API Name	**Return Type**	**Description**
pa_GetHandle	paHandleT	Returns a handle of an object corresponding to the given attribute
pa_GetHandleByName	paHandleT	Returns a handle of another object matching a given name
pa_GetInt	int	returns the integer value of the given attribute
pa_GetStr	char*	returns the string value of the given attribute
pa_GetSeqIterator	paIteratorT	returns the iterator to list of objects for the given attribute
pa_GetNext	paHandleT	returns the next handle in a given iterator
pa_GetHandleType	int	returns the object type of a given handle
pa_IsInClass	int	Returns 1 if object belongs to specified class else 0

pa_GetHierPathname	char*	returns the hierarchical path of a given object
pa_IsSameHandle	int	Returns 1 if handles are same else 0

Table 1: C PA-API

B. Tcl API for PA-IM

Tcl PA-API	
API Name	**Description**
pa_get_object_handle_by_name	Search object handle by name
pa_get_attribute	Get value of given attribute present on the handle
pa_get_hierpath	Return the full hierarchical pathname for specified handle
pa_is_in_class	Check if handle belongs to a specified class
pa_is_same_handle	Check if two handles are same

Table 2: Tcl PA-API

IX. EXAMPLES

In order to demonstrate the utility of the PA-IM and PA-APIs, we have used a sample design with three levels of hierarchy as shown below:

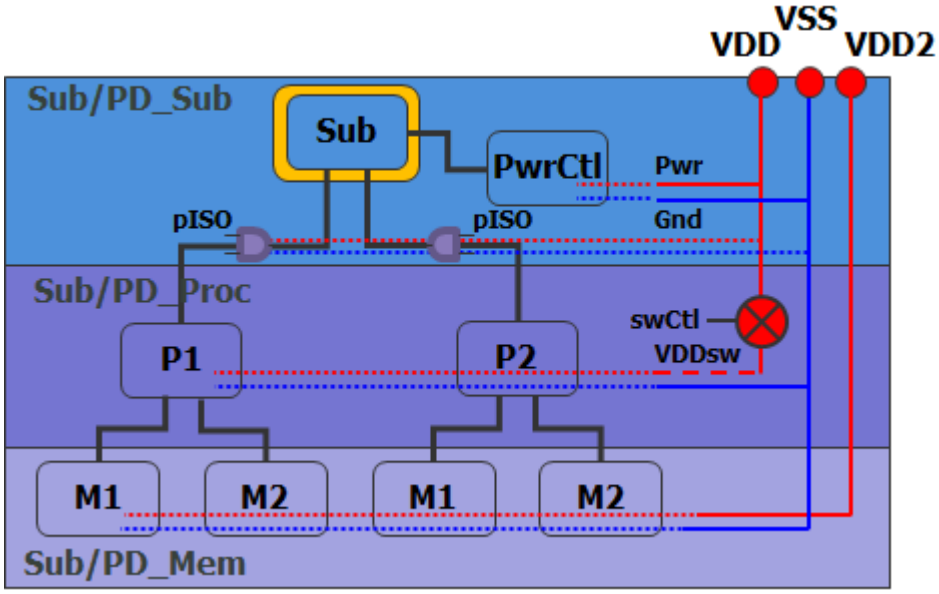

Fig 6. Design hierarchy with embedded PA objects

The power intent for the given hierarchy is captured in UPF file as shown below:

UPF File:

```
set_scope Sub
create_power_domain PD_Sub \
-include_scope
create_power_domain PD_Proc \
-elements {P1 P2}
create_power_domain PD_Mem \
-elements {P1/M1 P1/M2 P2/M1 P2/M2}
set_isolation ISOproc \
-domain PD_Proc \
```

```
-applies_to outputs \
-clamp_value 0 \
-location parent \
-isolation_power_net Pwr \
-isolation_ground_net Gnd
-isolation_signal pISO \
-isolation_sense high
...
```

Once the design goes through the UPF processing, it populates the PA-IM structure. An instance of the PA-IM structure is shown as UML Object diagram in the following figure:

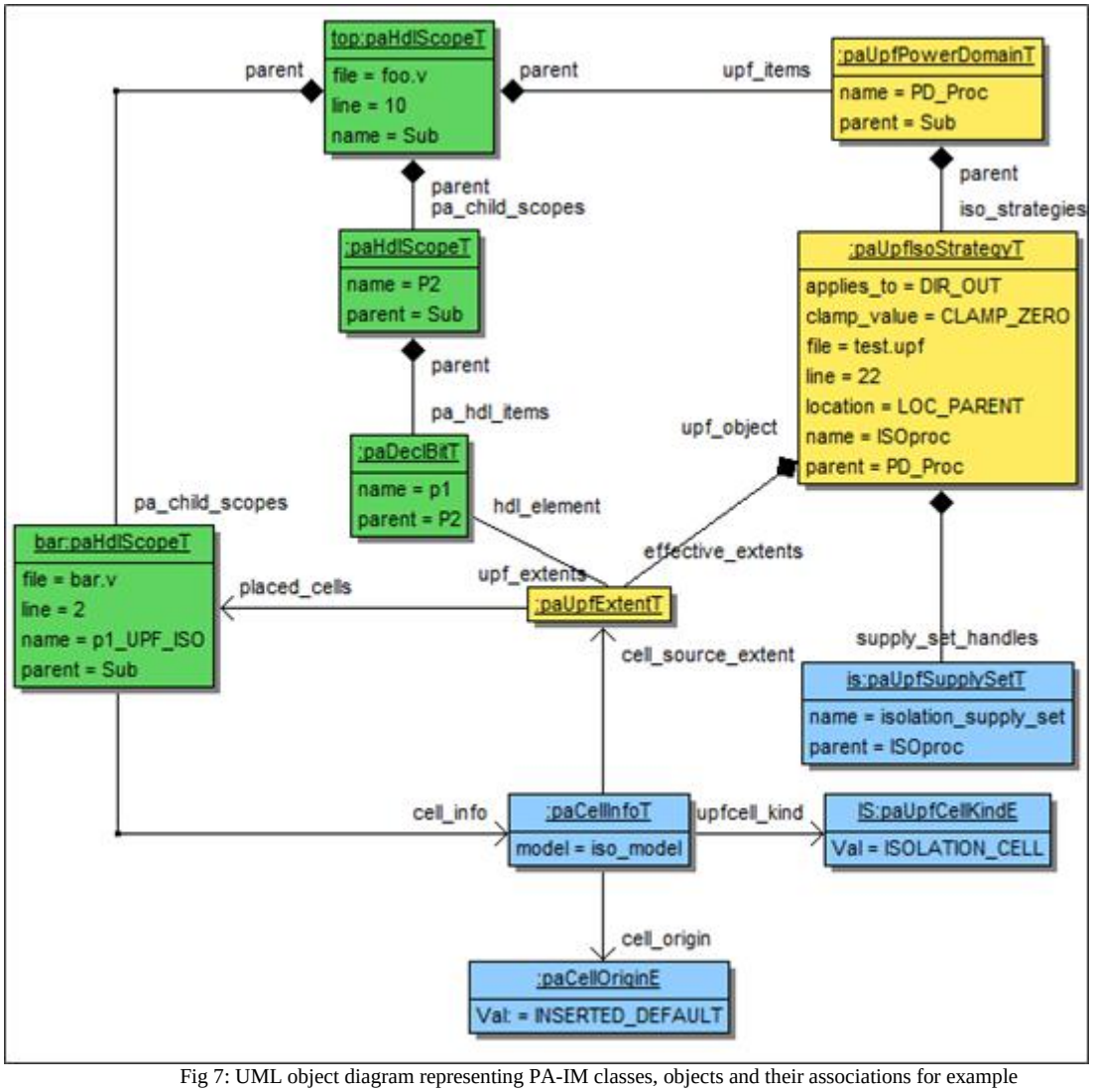

Fig 7: UML object diagram representing PA-IM classes, objects and their associations for example

Using the sample design and the populated PA-IM structures, we will demonstrate two specific kinds of PA-API usages. In the first example, we use C-API to access basic information about the Isolation Strategy and the ports it is isolating. In the second example, we use Tcl-API to print the list of power management cells within a given scope.

A. Print Isolation Information

C-Code:

```
pd = pa_GetHandleByName( NULL,"Sub/PD_Proc" );
iso_iter = pa_GetSeqIterator(pd, ISO_STRATEGIES);
pa_IterForeach( iso_iter, iso ) {
  pa_GetStr( iso, NAME );
 //=> "ISOproc"
  pa_GetInt(iso, CLAMP_VALUE, &clamp);
 //=> 0
  ctrl = pa_GetHandle(iso, ISOLATION_CONTROL);
  ctrl_sig = pa_GetHandle(ctrl, CONTROL_SIGNAL);
 //=> pISO
  pa_GetInt(ctrl, SIGNAL_SENSITIVITY, &ctrl_sense);
 //=> SENSE_HIGH
}
port_iter = pa_GetSeqIterator(iso, EFFECTIVE_EXTENTS);
pa_IterForeach(port_iter, port_ex) {
  port = pa_GetHandle(port_ex, HDL_ELEMENT);
  path = pa_GetFullPathname(port, NULL);
//=> /Sub/P2/p1
  cells_iter = pa_GetSeqIterator(port_ex, PLACED_CELLS);
  pa_IterForeach(cells_iter, cell) {
    path = pa_GetFullPathname(cell, NULL);
//=> /Sub/p1_UPF_ISO
  }
}
```

B. *Return a Tcl List of All Isolation Cells in a scope*

Tcl-Code:

```
proc get_iso_cells {scope}{
  set cells {}
# Check if Valid Handle
  if{[pa_is_in_class -handle scope -class_id CLASS_HDL_SCOPE] == 0} {
    return $cells
  }
# Get Child Scopes
  set children [pa_get_attribute -handle $scope -attribute CHILD_SCOPES ]
# Iterate over children and check if isolation
  foreach ch $children {
    set cell [pa_get_attribute -handle $ch -attribute CELL_INFO]
    if { $cell ne "" } {
      set cell_type [pa_get_attribute -handle $cell -attribute UPFCELL_KIND]
      if { $cell_type == ISOLATION_CELL } {
        lappend cells [pa_get_hierpath $cell]
      }
    } else {
      concat $cells [get_iso_cells $cell]
    }
  }
  return $cells
}
puts [get_iso_cell [pa_get_object_handle_by_name -name "/Sub"]]
# => { /Sub/p1_UPF_ISO }
```

X. Limitations and Future Work

The following are some of the limitations of the current work:

- Representing dynamic information for creating testbenches.

- Allowing an interface for modeling coverage metrics of Power management objects
- Capturing the connectivity information to address driver/receiver supply information.

XI. Conclusion

The PA-IM provides a well-defined structure to power management architecture. Along with PA-APIs it provides a simplified access to such information which can be used across tools and design flows. Because it captures the result of the application of power intent and stores the abstract representation of HDL, it can represent the power management information at different levels of design abstraction – RTL or Gate Level. It adds a new dimension to the verification and will empower tools to provide easy access to PA information.

The IEEE P1801 UPF working group has recognized the importance of an information model for UPF and has formed a sub-committee to work on the information model for UPF. The present work has already been donated in the IEEE P1801 UPF working group and is currently being studied for standardization.

References

[1] IEEE Std 1801™-2013 for Design and Verification of Low Power Integrated Circuits. IEEE Computer Society, 29 May 2013.
[2] Tina Lee (1999). "Information modeling from design to implementation", National Institute of Standards and Technology.
[3] Unified Modeling Language, http://www.uml.org
[4] UML Class Diagrams, http://en.wikipedia.org/wiki/Class_diagram
[5] UML Object Diagrams, http://en.wikipedia.org/wiki/Object_diagram