The Global Activity Format and its Applications

Version 0.0
February 11, 2002

This document describes the purpose and contents of the Global Activity Format (GAF) and its applications.

1.0 Purpose

The GAF describes instance-specific switching activity information of a digital integrated circuit. This information can be extracted from simulation, or it can be estimated using probabilistic analysis.

The information contained in the GAF file can be used for target applications including, but not restricted to, power analysis, electromigration analysis, delay and crosstalk test.

In terms of information detail, a GAF file is positioned in-between a Value Change Dump (VCD) file and a Switching Activity Format (SAF) file.

A VCD file reports all simulation events. Therefore it is suitable for applications which need detailed information about simulation events. Many applications do not need this level of detail. The VCD is also not suitable for applications without simulation.

The SAF reports nodal switching activity. This information is suitable for applications such as rough power estimation or for static fault coverage. However, IC libraries may provide more accurate models, for example arc-or vector-specific power consumption. These models cannot be accurately evaluated, if only nodal switching activity is given for the design.

The GAF describes correlated switching activity information which corresponds to the models provided in the IC library. For example, if a library contains arc-or vector-specific power models, the GAF file reports the switching activity of the arc or vector. Arcs or vectors can be used to describe characteristics other than power consumption, for example electromigration, delay etc. Therefore the GAF is suitable in the context of many applications for evaluation of such models. The Advanced Library Format (ALF) provides the matching description of arc-or vector-specific models referred in GAF.

2.0 Contents of a GAF file

2.1 Formal definition

A GAF file consists of a header section and a data section.

The header section contains information such as tool version, total simulation time etc. For details please see [1].

The data section contains node data and vector data.

```
node_data ::=
   hierarchical_terminal state_probability rise_count fall_count

vector_data ::=
   hierarchical_vector state_probability activity_count

hierarchical_terminal ::=
   module_name { . instance_name } . terminal_name

hierarchical_vector ::=
   module_name { . instance_name } . control_expression

control_expression ::=
   ( vector_expression | boolean_expression )
```

where `state_probability` is a number between 0 and 1, `rise_count`, `fall_count`, and `activity_count` are positive integers, `vector_expression` and `boolean_expression` are defined in the ALF standard [2].

The `node_data` is equivalent to the data in a SAF. The `vector_data` is a unique feature of GAF.

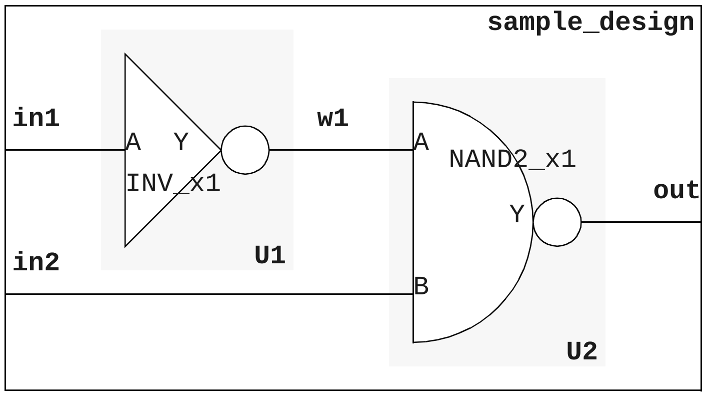
2.2 Example

The following example explains the contents of the GAF file. Required related information is a design netlist and a library.

A sample design netlist described in Verilog is given below:

```
module sample_design (in1, in2, out);
   input in1;
   input in2;
   output out;
   wire w1;
   INV_x1 U1 (.A(in1),.Y(w1));
   NAND2_x1 U2 (.A(w1),.B(in2),.Y(out));
endmodule
```

The netlist is equivalent to the following schematic:

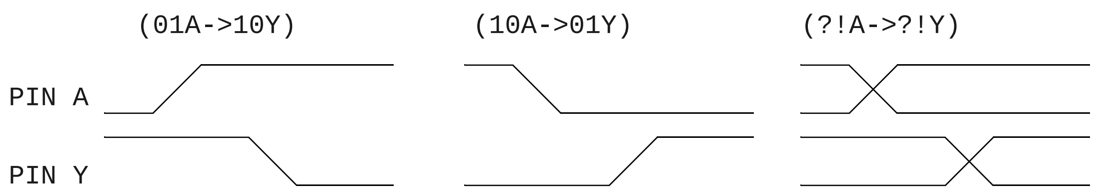


This design instantiates the library elements `INV_x1` and `NAND2_x1`. These cells are described in ALF as follows:

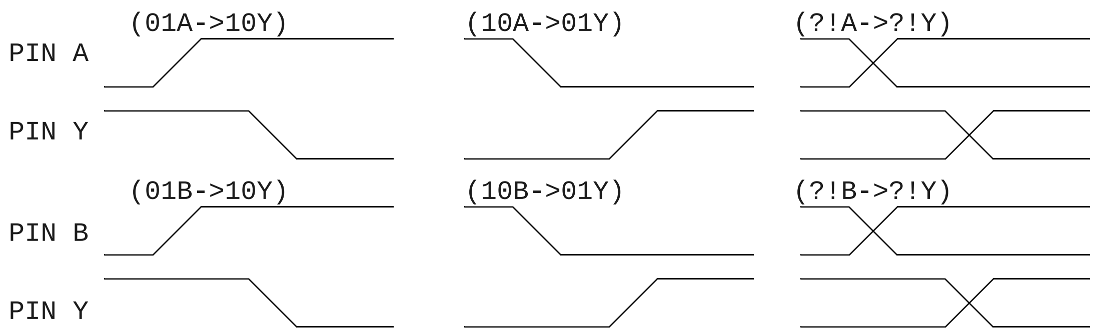
```
LIBRARY sample_library {
   CELL INV_x1 {
      PIN A { DIRECTION = input; }
      PIN Y { DIRECTION = output; }
      FUNCTION { BEHAVIOR { Y = !A; } }
      VECTOR ( 01 A -> 10 Y ) {
         DELAY { FROM { PIN=A; } TO { PIN=Y; } /* put data here */ }
         ENERGY { /* put data here */ }
      }
      VECTOR ( 10 A -> 01 Y ) {
         DELAY { FROM { PIN=A; } TO { PIN=Y; } /* put data here */ }
         ENERGY { /* put data here */ }
      }
      VECTOR ( ?! A -> ?! Y ) {
         LIMIT { FREQUENCY { MAX { /* put data here */ } } }
      }
   }
   CELL NAND2_x1 {
      PIN A { DIRECTION = input; }
      PIN B { DIRECTION = input; }
      PIN Y { DIRECTION = output; }
      FUNCTION { BEHAVIOR { Y = !(A && B); } }
      VECTOR ( 01 A -> 10 Y ) {
         DELAY { FROM { PIN=A; } TO { PIN=Y; } /* put data here */ }
         ENERGY { /* put data here */ }
      }
      VECTOR ( 10 A -> 01 Y ) {
         DELAY { FROM { PIN=A; } TO { PIN=Y; } /* put data here */ }
         ENERGY { /* put data here */ }
      }
      VECTOR ( 01 B -> 10 Y ) {
         DELAY { FROM { PIN=B; } TO { PIN=Y; } /* put data here */ }
         ENERGY { /* put data here */ }
      }
      VECTOR ( 10 B -> 01 Y ) {
         DELAY { FROM { PIN=B; } TO { PIN=Y; } /* put data here */ }
         ENERGY { /* put data here */ }
      }
      VECTOR ( ?! A -> ?! Y ) {
         LIMIT { FREQUENCY { MAX { /* put data here */ } } }
      }
      VECTOR ( ?! B -> ?! Y ) {
         LIMIT { FREQUENCY { MAX { /* put data here */ } } }
      }
   }
}
```

The vectors in the ALF model are equivalent to the following timing diagrams:

`sample_design.U1`



`sample_design.U2`



A corresponding GAF file may report the switching activities as follows:

```
sample_design.in1 0.4 9 8
sample_design.in2 0.5 4 4
sample_design.out 0.8 4 4
sample_design.w1  0.6 8 9
sample_design.U1.(01A->10Y) 0 9
sample_design.U1.(10A->01Y) 0 8
sample_design.U1.(?!A->?!Y) 0 17
sample_design.U2.(01A->10Y) 0 1
sample_design.U2.(10A->01Y) 0 1
sample_design.U2.(?!A->?!Y) 0 2
sample_design.U2.(01B->10Y) 0 3
sample_design.U2.(10B->01Y) 0 3
sample_design.U2.(?!B->?!Y) 0 6
```

The `node_data` in this example are the following:

```
sample_design.in1 0.4 9 8
sample_design.in2 0.5 4 4
sample_design.out 0.8 4 4
sample_design.w1  0.6 8 9
```

They must match the design netlist.

The `vector_data` in this example are the following:

```
sample_design.U1.(01A->10Y) 0 9
sample_design.U1.(10A->01Y) 0 8
sample_design.U1.(?!A->?!Y) 0 17
sample_design.U2.(01A->10Y) 0 1
sample_design.U2.(10A->01Y) 0 1
sample_design.U2.(?!A->?!Y) 0 2
sample_design.U2.(01B->10Y) 0 3
sample_design.U2.(10B->01Y) 0 3
sample_design.U2.(?!B->?!Y) 0 6
```

They must match both the design netlist and the ALF library.

Note the following particularities:

The dot is the only legal hierarchy separator. There is no leading dot.

The `control_expression` in GAF has no whitespace, whereas the corresponding `control_expression` in ALF may have whitespace. The purpose is to allow identification of the `control_expression` by simple pattern matching.

The `state_probability` for `vector_data` in case of a `vector_expression` is always 0. The `state_probability` in case of a `boolean_expression` is in the range between 0 and 1.

The relevance of the `vector_data` can be easily seen. Library data, such as `DELAY` and `ENERGY`, is associated with instances of a `VECTOR`, such as `(01A->10Y)`, `(10A->01Y)`, `(01B->10Y)`, and `(10B->01Y)`, for the instance `sample_design.U2` of `NAND2_x1`. The `node_data` does not specify how often a particular instance of a `VECTOR` is activated.

In this particular example, the node `sample_design.in1` is more active than the node `sample_design.in2`, however, the vector `sample_design.U2.(?!B->?!Y)` is more active than the vector `sample_design.U2.(?!A->?!Y)`. In general, the activity of a vector can not be inferred by the activity of a node related to a vector.

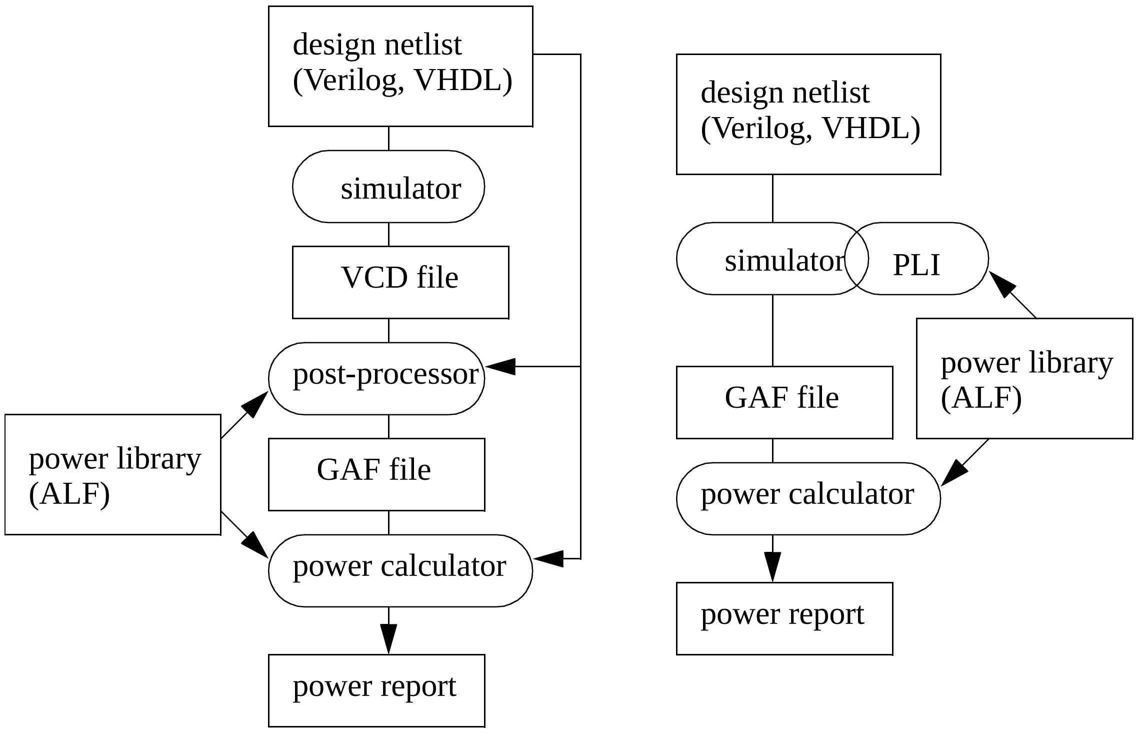
Therefore, in a power analysis application, the `vector_data` must be reported to enable accurate power calculation. In a delay test application, the `vector_data` must be reported to show that the actual timing arc under test is activated.

3.0 Applications for GAF

3.1 Event-driven simulation and power calculation

The GAF can be used as an output of a value-change-dump (VCD) postprocessor. Alternatively, a PLI can be linked to the simulator to generate a GAF file directly.

FIGURE 1. Design flow involving GAF for simulation and power calculation



The GAF file is subsequently used as input to a power calculation tool. This is a mainstream application today.

Example:

```
sample_design.U1.(01A->10Y) 0 9
sample_design.U1.(10A->01Y) 0 8
sample_design.U2.(01A->10Y) 0 1
sample_design.U2.(10A->01Y) 0 1
sample_design.U2.(01B->10Y) 0 3
sample_design.U2.(10B->01Y) 0 3
```

From the ALF library, the following energy consumption values are calculated:

3 pJ for `sample_design.U1.(01A->10Y)`.
12 pJ for `sample_design.U1.(10A->01Y)`.

4 pJ for `sample_design.U2.(01A->10Y)`.
15 pJ for `sample_design.U2.(10A->01Y)`.
5 pJ for `sample_design.U2.(01B->10Y)`.
14 pJ for `sample_design.U2.(10B->01Y)`.

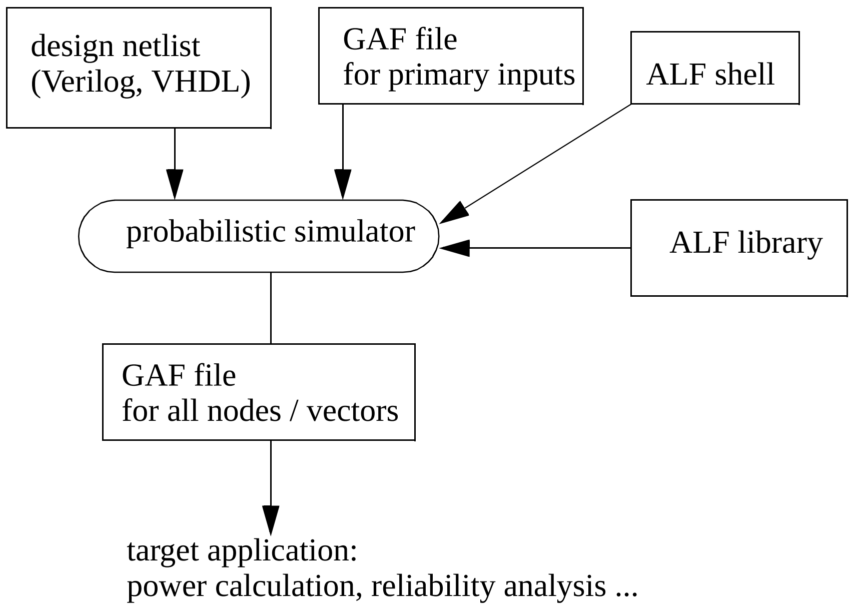
Therefore the total energy consumption is
3 pJ * 9 + 12 pJ * 8 + 4 pJ * 1 + 15 pJ * 1 + 5 pJ * 3 + 14 pJ * 3 = 199 pJ.

3.2 Probabilistic simulation and reliability analysis

The GAF can be used as an output of a probabilistic simulator. In this case, the node and vector switching activities are estimated rather than simulated.

It is also conceivable to use the GAF as an input of a probabilistic simulator. A probabilistic simulator needs switching activity specification for primary inputs of a circuit under analysis. One advantage of GAF is the capability of describing correlated events, since a `control_expression` can express a state, an event, a sequence of events, simultaneous events, each of which can have a state condition.

FIGURE 2. Design flow involving GAF for probabilistic simulation



The GAF file can then be used as input to a target application, such as power calculation or reliability analysis, in particular signal electromigration. The probabilistic simulator may have some built-in pessimism to make sure that the switching frequency is not underestimated in any case. The `node_data` within the GAF file is usable for interconnect electromigration check. The `vector_data` is usable for cell electromigration check.

Example:

```
   sample_design.U1.(?!A->?!Y) 0 17
   sample_design.U2.(?!A->?!Y) 0 2
   sample_design.U2.(?!B->?!Y) 0 6
```

The simulation time is 20 ns. Therefore, the actual vector frequencies are as follows:

850 MHz (17 / 20 ns) for `sample_design.U1.(?!A->?!Y)`.
100 MHz (2 / 20 ns) for `sample_design.U1.(?!A->?!Y)`.
300 MHz (6 / 20 ns) for `sample_design.U1.(?!A->?!Y)`.

From the ALF library, the following vector frequency limits are calculated.

800 MHz for `sample_design.U1.(?!A->?!Y)`.
400 MHz for `sample_design.U2.(?!A->?!Y)`.
500 MHz for `sample_design.U2.(?!B->?!Y)`.

Therefore `sample_design.U1.(?!A->?!Y)` violates the frequency limit.

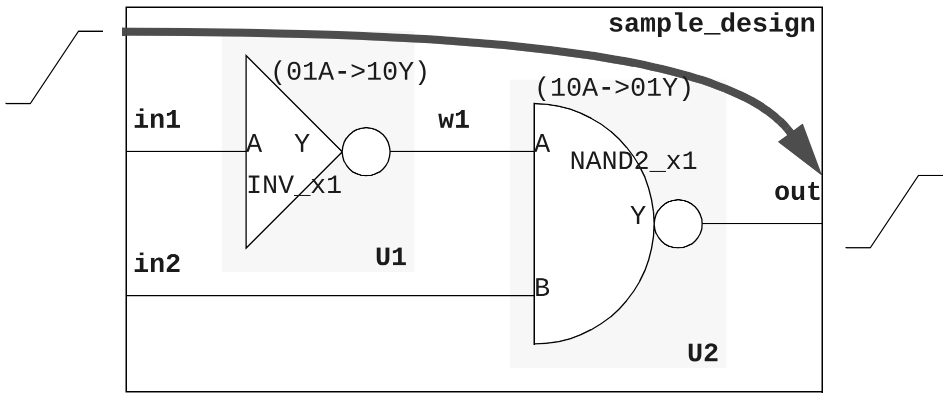
3.3 Delay and crosstalk testing

The purpose of delay testing is to find a simulation pattern which activates a path subject to delay measurement on a tester. In case of crosstalk testing, both an aggressor and a victim path must be activated simultaneously.

A GAF file can report the switching activity within a single clock cycle. Such a GAF file can be inspected for the purpose to check whether all the relevant delay vectors along the path have been activated.

An example is shown below.

FIGURE 3. Illustration of GAF usage for delay test



In order to activate the path from rising edge at **`in1`** to rising edge at **`out`**, the vector instances `sample_design.U1.(01A->10Y)` and `sample_design.U2.(10A->01Y)` must be activated within the same clock cycle.

4.0 Conclusion

The GAF is useful for many applications, such as reporting simulation-based vector switching activity for power calculation, describing correlated switching activity for probabilistic simulation targeted for power calculation and/or reliability analysis, reporting activity of timing vectors along a path under test.

GAF is closely related to a cell model described in ALF, in a similar way as VCD is closely related to a design netlist described in Verilog or VHDL. Therefore we propose to deploy GAF as a companion standard to ALF.

References

[1] Sequence documentation on GAF

[2] ALF version 2.0