Library Harmonization for Power

Version	Date
0.0	2004/03/22
0.1	2004/05/01
0.2	2004/06/23
0.3	2004/10/10

Table of contents

1.0 Electrical Power

For CMOS circuits, power can be divided into subcategories: leakage, switching, internal.

- Switching power is associated with charge/discharge of external load capacitance. As it is a transient phenomenon, it is accounted for as energy rather than power.
- Internal power is associated with short-circuit current and charge/discharge of internal capacitances. As it is a transient phenomena, it is accounted for as energy rather than power.
- Leakage power is dependent on transistor threshold voltages. Voltage and current are constant during the measurement.

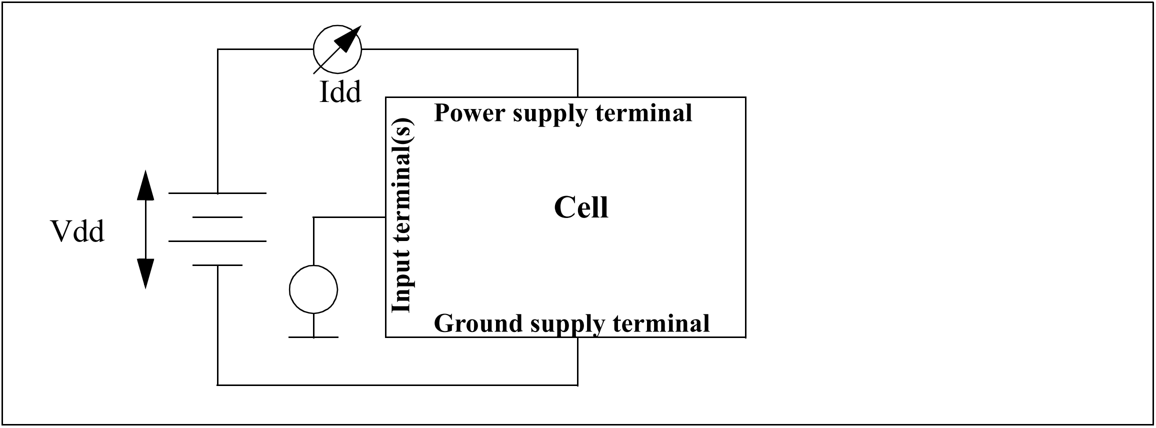
Switching and internal power, and to a lesser degree leakage power, are state-dependent.

In remainder of this section, a single constant power supply is assumed. Multiple power supplies are addressed in the next section. The voltage levels of signals in CMOS circuits are given by the electrical potentials of the power and ground supply.

1.1 Leakage power

Leakage power is measured by applying constant input voltages and a static current meter in series to the supply voltage to a cell, as illustrated in Figure 1 on page 3. The output pins are left unconnected.

FIGURE 1. Measurement of leakage power



The leakage power is given by the product Idd*Vdd.

Leakage power can be state-dependent or state-independent. The following Figure 2 on page 4 shows the description of both state-independent and state-dependent leakage power in Liberty.

FIGURE 2. Leakage power description in Liberty

```
/* Liberty */
cell (CellName) {
  cell_leakage_power : CellLeakagePowerValue ;
  leakage_power () {
    when : "BooleanExpression"
    value : LeakagePowerValue ;
  }
}
```

The following Figure 3 on page 4 shows the description of leakage power in ALF.

FIGURE 3. Leakage power description in ALF

```
/* ALF */
CELL CellName {
  POWER = CellLeakagePowerValue {
    MEASUREMENT = static ;
    CALCULATION = ABSOLUTE | INCREMENTAL ;
  }
  VECTOR (BooleanExpression) {
    POWER = LeakagePowerValue { MEASUREMENT = static ; }
  }
}
```

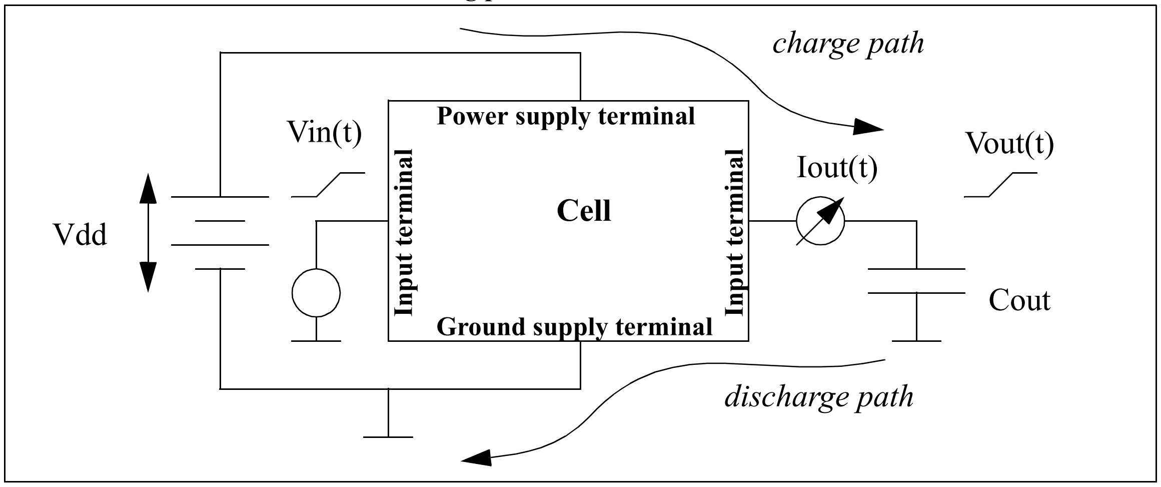
The `CALCULATION` annotation specifies whether or not the state-independent *`CellLeakagePowerValue`* includes the state-dependent *`LeakagePowerValue`*. An **`INCREMENTAL`** value shall be added to a state-dependent value. An **`ABSOLUTE`** value shall be used instead of a state-dependent value.

See also Section 1.4, “Power with state-dependency,” on page 8.

1.2 Switching power

Transient power is dissipated when a transient voltage Vin(t) is applied to an input terminal that causes an output terminal to switch while the output terminal is connected to a capacitor Cout, as illustrated in Figure 4 on page 5. The energy transferred between the cell’s power supply and the capacitor is called switching energy.

FIGURE 4. Measurement of switching power



The energy stored in the capacitor is given by the integral over Iout(t)*Vout(t), where Iout(t) = Cout * dVout(t)/dt.

This integral evaluates to Cout * 1/2 * ($Vout^2$(t=T) - $Vout^2$(t=0)), where T is the time it takes to completely charge or discharge Cout .

When the output signal rises, i.e., Vout(t) changes from 0 to Vdd, Iout(t) flows from the power supply terminal to Cout (see *charge path*). Therefore the supplied energy is given by the integral over Iout(t)*Vdd.

When the output signal falls, i.e., Vout(t) changes from Vdd to 0, Iout(t) flows from Cout to the ground supply terminal (see *discharge path*). Therefore the supplied energy is zero.

Given this information, the energy components can be mathematically evaluated, as shown in the following table.

TABLE 1. Evaluation of switching energy

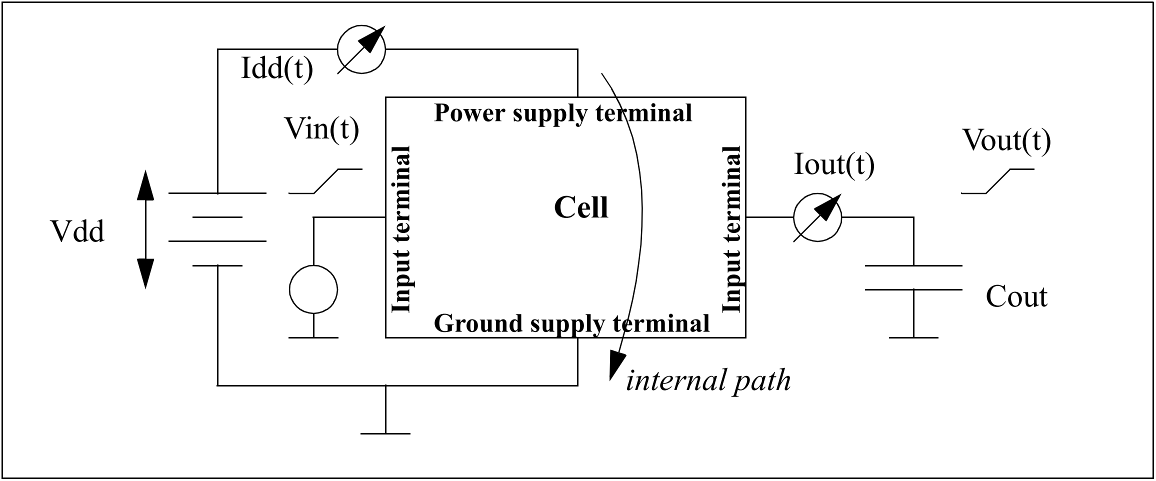
quantity	**calculation**	**output rising**	**output falling**
initial energy stored in Cout	$E_{initial}$ = 1/2*Cout*$Vout^2$(t=0)	0	1/2*Cout*Vdd^2
final energy stored in Cout	E_{final} = 1/2*Cout*$Vout^2$(t=T)	1/2*Cout*Vdd^2	0
energy supplied through cell	E_{supply}	Cout*Vdd^2	0
energy dissipated in the cell	E_{supply} - (E_{final} - $E_{initial}$)	1/2*Cout*Vdd^2	1/2*Cout*Vdd^2

In a supply-oriented accounting method, the switching energy is simply given by the supplied energy. In a dissipation-oriented accounting method, the switching energy is given by the difference between the supplied energy and the stored energy in the capacitor. Regardless of which accounting method is used, the switching energy for output rising and output falling adds always up to Cout*Vdd^2. Therefore no library characterization is needed for mathematical evaluation of the switching energy per se.

1.3 Internal power

Transient energy supplied and eventually dissipated in a cell is measured by applying a transient current meter in series to the power supply terminal of the cell, as illustrated in Figure 5 on page 6. The total transient energy is given by the integral over Vdd*Idd(t). The internal energy is the difference between the total transient energy and the switching energy (see Section 1.2 on page 4).

FIGURE 5. Measurement of internal power



The following Figure 6 on page 6 shows the description of transient energy in Liberty.

FIGURE 6. Transient energy description in Liberty

```
/* Liberty */
cell (CellName) {
  pin(RelatedPinName) {
    direction : RelatedPinDirection;
  }
  pin(PinName) {
    direction : PinDirection;
    internal_power() {
      related_pin  : "RelatedPinName"; /* this is optional */
      /* lib_PowerModel */
      ModelKeyword (CalculationType) { values ( /* lib_Data */ ); }
    }
}
```

The following Figure 7 on page 7 shows the description of transient energy in ALF.

FIGURE 7. Transient energy description in ALF

```
/* ALF */
CELL CellName {
  PIN RelatedPinName {
    DIRECTION = RelatedPinDirection;
  }
  PIN PinName {
    DIRECTION = PinDirection;
  }
  VECTOR (VectorExpression) {
    ENERGY {
      /* ALF_data */
    }
  }
}
```

The ModelKeyword in Liberty specifies whether the transition observed at the pin associated with the transient energy is rising or falling. In ALF, the transition is specified within the vector expression.

The mapping between Liberty and ALF is shown in the following Table 2 on page 7.

TABLE 2. Mapping of Liberty and ALF constructs for transient energy measurements

Liberty construct yes = with related pin no = without related pin		**ALF construct** PN = Pin Name RPN = Related Pin Name	**Comment**
Model Keyword		***Vector Expression***	see note [a]
power	no	?! PN	model applies for any transition
rise_power	no	01 PN	model applies for rise transition
fall_power	no	10 PN	model applies for fall transition
power	yes	?! RPN -> ?! PN	model applies for any transition
rise_power	yes	?! RPN -> 01 PN	model applies for rise transition
fall_power	yes	?! RPN -> 10 PN	model applies for fall transition

a. Note: The transition of the related pin (rise or fall) can not be expressed explicitly in Liberty. It has to be derived from the logical function and/or the unateness of a corresponding timing arc.

In Liberty, the data for transient energy measurements represents only the internal energy. In ALF, the data represents the total transient energy. To convert from Liberty to ALF data, one of the following methods can be used:

1. Add the switching energy data 1/2*Cout*Vdd^2 to every “power” model. Add either Cout*Vdd^2 to “rise_power” or 1/2*Cout*Vdd^2 to both “rise_power” and “fall_power”.

2. Introduce one extra power vector (?! PN) or two extra power vectors (01 PN) and (10 PN) with energy data 1/2*Cout*Vdd^2 for each.
   Alternatively, add just one extra power vector (01 PN) with energy data Cout*Vdd^2.

To convert from ALF to Liberty, subtract 1/2*Cout*Vdd^2 from every power vector featuring the sub-expression (... ?! PN). Subtract 1/2*Cout*Vdd^2 also from every power vector featuring one of the sub-expressions (... 01 PN) or (10 PN). Alternatively, subtract Cout*Vdd^2 from (... 01 PN) only, subtract nothing from (10 PN). If the resulting energy data evaluates to zero, then the power vector can be eliminated altogether.

1.4 Power with state-dependency

State-dependent energy is expressed using the “when” statement in Liberty and a boolean co-factor in the vector expression in ALF. An example is shown below.

FIGURE 8. State-dependent energy example in Liberty and ALF

```
/* Liberty */                      /* ALF */
pin(Y) {
  internal_power() {               VECTOR ((?! A -> 01 Y)&(E1&E2)) {
    related_pin : "A";               ENERGY ...
    when : "E1&E2";                }
    rise_power ...                 VECTOR ((?! A -> 10 Y)&(E1&E2)) {
    fall_power ...                   ENERGY ...
  }                                }
  internal_power() {               VECTOR ((?! A -> 01 Y)&(E1&!E2)) {
    related_pin : "A";               ENERGY ...
    when : "E1&!E2";               }
    rise_power ...                 VECTOR ((?! A -> 10 Y)&(E1&!E2)) {
    fall_power ...                   ENERGY ...
  }                                }
  internal_power() {               VECTOR ((?! A -> 01 Y)&(!E1&E2)) {
    related_pin : "A";               ENERGY ...
    when : "!E1&E2";               }
    rise_power ...                 VECTOR ((?! A -> 10 Y)&(!E1&E2)) {
    fall_power ...                   ENERGY ...
  }                                }
}
```

A default condition in Liberty is specified by the absence of a “when” statement.

A default value condition in ALF is specified by a power model with **`CALCULATION`** annotation value **`absolute`** in the context of an ALF vector without boolean co-factor[1].

1. If the value of the **`CALCULATION`** annotation is **`absolute`**, then the power data shall not be combined with any other power data. If the value of the **`CALCULATION`** annotation is **`incremental`**, then the power data can be combined with other power data.

The default condition shall apply in the following situations:

- None of the specified value conditions evaluates true
- The application context (e.g. a record of simulation events) does not provide enough information to decide whether one, and only one, specified value condition evaluates true
- The application tool does not support evaluation of a value condition

1.5 Power involving a bus or a bundle of pins

The following power relationships involving multiple pins can be defined in Liberty:

TABLE 3. power relationships involving multiple pins

object	related object	comment
scalar pin	scalar pin	described in Section 1.3 on page 6
scalar pin	multiple scalar pins	see Figure 9 on page 10
scalar pin	bus	not in Liberty doc
bundle	scalar pin	see Figure 10 on page 10
bundle	bitwidth-compatible bundle	see Figure 11 on page 10
bundle	multiple scalar pins and/or bitwidth-compatible bundle	see Figure 12 on page 11
bundle	bus	not in Liberty doc
bus	scalar pin	see Figure 13 on page 12
bus	bitwidth-compatible bus	see Figure 14 on page 12
bus	multiple scalar pins and/or bitwidth-compatible bus	see Figure 15 on page 13
bus	bus	see Figure 16 on page 14

FIGURE 9. Power relationship between scalar pin and multiple scalar pins

```
/* Liberty */                       /* ALF */
pin(A) { direction : input ; }      PIN A { DIRECTION = input; }
pin(B) { direction : input ; }      PIN B { DIRECTION = input; }
pin(C) { direction : input ; }      PIN C { DIRECTION = input; }
pin(Y) { direction : output ;       PIN Y { DIRECTION = output; }
  internal_power() {
    related_pin : "A B C";          GROUP A_B_C { A B C }
    power ...                       VECTOR (?! A_B_C -> ?! Y) {
  }                                   ENERGY ...
}                                   }
```

FIGURE 10. Power relationship between bundle and single scalar pin

```
/* Liberty */                       /* ALF */
pin(A) { direction : input ; }      PIN A { DIRECTION = input; }
pin(X) { direction : output ; }     PIN X { DIRECTION = output; }
pin(Y) { direction : output ; }     PIN Y { DIRECTION = output; }
pin(Z) { direction : output ; }     PIN Z { DIRECTION = output; }
bundle(X_Y_Z) {
  members (X,Y,Z);                  GROUP X_Y_Z { X Y Z }
  internal_power() {                VECTOR (?! A -> ?! X_Y_Z) {
    related_pin : "A";                ENERGY ...
    power ...                       }
  }
}
```

FIGURE 11. Power relationship between bundle and bitwidth-compatible bundle

```
/* Liberty */                       /* ALF */
pin(A) { direction : input ; }      PIN A { DIRECTION = input; }
pin(B) { direction : input ; }      PIN B { DIRECTION = input; }
pin(C) { direction : input ; }      PIN C { DIRECTION = input; }
pin(X) { direction : output ; }     PIN X { DIRECTION = output; }
pin(Y) { direction : output ; }     PIN Y { DIRECTION = output; }
pin(Z) { direction : output ; }     PIN Z { DIRECTION = output; }
bundle(A_B_C) {                     PINGROUP[1:3] A_B_C {
  members (A,B,C);                    MEMBERS { A B C }
}                                   }
bundle(X_Y_Z) {                     PINGROUP[1:3] X_Y_Z {
  members (X,Y,Z);                    MEMBERS { X Y Z }
  internal_power() {                }
    related_pin : "A_B_C";          GROUP BundleBit { 1 : 3 }
    power ...                       VECTOR (
  }                                   ?! A_B_C[BundleBit] ->
}                                     ?! X_Y_Z[BundleBit]) {
                                      ENERGY ...
                                    }
```

FIGURE 12. Power relationship between bundle and multiple scalar pins and/or bitwidth-compatible bundles

```
/* Liberty */
pin(A) { direction : input ; }
pin(B) { direction : input ; }
pin(C) { direction : input ; }
pin(D) { direction : input ; }
pin(E) { direction : input ; }
pin(X) { direction : output ; }
pin(Y) { direction : output ; }
pin(Z) { direction : output ; }
bundle(A_B_C) {
  members (A,B,C);
}
bundle(X_Y_Z) {
  members (X,Y,Z);
  internal_power() {
    related_pin : "A_B_C D E";
    power ...
  } /* equivalent to following:
  internal_power() {
    related_pin : "A_B_C";
    power ...
  }
  internal_power() {
    related_pin : "D E";
    power ...
  } */
}
```

```
/* ALF */
PIN A { DIRECTION = input; }
PIN B { DIRECTION = input; }
PIN C { DIRECTION = input; }
PIN D { DIRECTION = input; }
PIN E { DIRECTION = input; }
PIN X { DIRECTION = output; }
PIN Y { DIRECTION = output; }
PIN Z { DIRECTION = output; }
PINGROUP[1:3] A_B_C {
  MEMBERS { A B C }
}
PINGROUP[1:3] X_Y_Z {
  MEMBERS { X Y Z }
}
GROUP BundleBit { 1 : 3 }
GROUP D_E { D E }
VECTOR (
  ?! A_B_C[BundleBit] ->
  ?! X_Y_Z[BundleBit]) {
  ENERGY ...
}
VECTOR (
  ?! D_E ->
  ?! X_Y_Z[BundleBit]) {
  ENERGY ...
}
```

FIGURE 13. Power relationship between bus and single scalar pin

```
/* Liberty */                       /* ALF */
  type(Bit) {                       GROUP Bit { 1 : 3 }
    base_type : array ;
    data_type : bit ;
    bit_width : 3 ;
    bit_from : 1 ;
    bit_to : 3 ;
    downto : false ;
  }
pin(A) { direction : input ; }      PIN A { DIRECTION = input; }
bus(Y) {                            PIN [1:3] Y { DIRECTION=output; }
  direction : output ;
  bus_type : Bit ;                  VECTOR (?! A -> ?! Y[Bit]) {
  internal_power() {                  ENERGY ...
    related_pin : "A";              }
    power ...
  }
}
```

FIGURE 14. Power relationship between bus and bitwidth-compatible bus

```
/* Liberty */                       /* ALF */
  type(Bit) {                       GROUP Bit { 1 : 3 }
    base_type : array ;
    data_type : bit ;
    bit_width : 3 ;
    bit_from : 1 ;
    bit_to : 3 ;
    downto : false ;
  }
bus(A) {                            PIN [1:3] A { DIRECTION = input; }
  direction : input ;
  bus_type : Bit ;
}
bus(Y) {                            PIN [1:3] Y { DIRECTION=output; }
  direction : output ;
  bus_type : Bit ;
  internal_power() {
    related_pin : "A";              VECTOR (?! A[Bit] -> ?! Y[Bit]) {
    power ...                         ENERGY ...
  }                                 }
}
```

FIGURE 15. Power relationship between bus and multiple scalar pins and/or bitwidth-compatible bus

```
/* Liberty */                      /* ALF */
  type(Bit) {                      GROUP Bit { 1 : 3 }
    base_type : array ;
    data_type : bit ;
    bit_width : 3 ;
    bit_from : 1 ;
    bit_to : 3 ;
    downto : false ;
  }
bus(A) {                           PIN [1:3] A { DIRECTION = input; }
  direction : input ;
  bus_type : Bit ;
}
pin(B) { direction : input ; }     PIN B { DIRECTION = input; }
pin(C) { direction : input ; }     PIN C { DIRECTION = input; }
bus(Y) {                           PIN [1:3] Y { DIRECTION=output; }
  direction : output ;
  bus_type : Bit ;
  internal_power() {
    related_pin : "A B C";
    power ...
  } /* equivalent to following:
  internal_power() {               VECTOR (?! A[Bit] -> ?! Y[Bit]) {
    related_pin : "A";               ENERGY ...
    power ...                      }
  }
  internal_power() {               GROUP B_C { B C }
    related_pin : "B_C";           VECTOR (?! B_C -> ?! Y[Bit]) {
    power ...                        ENERGY ...
  } */                             }
}
```

FIGURE 16. Power relationship between bus and bus with arbitrary bitwidth

```
/* Liberty */                          /* ALF */
  type(Bit) {                          GROUP Bit { 1 : 3 }
    base_type : array ;                GROUP OtherBit { 1 : 3 }
    data_type : bit ;
    bit_width : 3 ;
    bit_from : 1 ;
    bit_to : 3 ;
    downto : false ;
  }
bus(A) {
  direction : input ;                  PIN [1:3] A { DIRECTION = input; }
  bus_type : Bit ;
}
bus(Y) {                               PIN [1:3] Y { DIRECTION=output; }
  direction : output ;
  bus_type : Bit ;                     VECTOR (
  internal_power() {                     ?! A[Bit] ->
    related_bus_pins : "A";              ?! Y[OtherBit]) {
    power ...                            ENERGY ...
  }                                    }
}
```

1.6 Power involving a switching interval

To do: Power involving a switching interval, rising|falling|switching_together

1.7 Power involving functionally equivalent output pins

To do: Power with equal_or_opposite output

1.8 Glitch power

Glitch is defined herein as the combination of two subsequent complementary events following each other in a short time interval. This time interval is shorter than the propagation delay of the first event. Therefore, the second event occurs before the effect of the first event can be fully observed.

In a simulator, A glitch occurs when a new transaction is scheduled at an absolute time which is greater than the absolute time of a previously scheduled pending event which results in a preemptive behavior.

2.0 Multiple power supplies

Multiple power supplies for a cell (e.g. level shifter, back-biasing) require each power supply voltage to become a characterization variable.

Association of rails with pins and with modes of power consumption is required in the library.

Liberty supports two variables for dual-supply voltage, voltage and voltage1. Also it supports a mapping construct between the variable and a power rail name.

```
variables(temperature, ..., voltage, voltage1) ;
mapping(voltage, VDD) ;
mapping(voltage1, VDD2) ;
```

The power_level attribute associates internal_power with a power rail.

FIGURE 17. Liberty description of association between energy and a power rail

```
/* Liberty */
library (LibraryName) {
  power_supply() {
    power_rail(VDD , 1.5) ;
    power_rail(VDD2 , 1.0) ;
  }
  cell (CellName) {
    pin(PinName) {
      direction : PinDirection;
      internal_power(E1) {
        power_level : VDD ;
        /* put lib_PowerModel here */
      }
      internal_power(E2) {
        power_level : VDD2 ;
        /* put lib_PowerModel here */
      }
    }
  }
}
```

FIGURE 18. ALF description of association between energy and a power rail

```
/* ALF */
LIBRARY LibraryName {
  CLASS VDD { USAGE = SUPPLY_CLASS ; VOLTAGE = 1.5 ; }
  CLASS VDD2 { USAGE = SUPPLY_CLASS ; VOLTAGE = 1.0 ; }
  CELL CellName {
    PIN PinName {
      DIRECTION = PinDirection ;
    }
    VECTOR (VectorExpression) {
      ENERGY E1 {
        SUPPLY_CLASS = VDD ;
        /* ALF_data */
      }
      ENERGY E2 {
        SUPPLY_CLASS = VDD2 ;
        /* ALF_data */
      }
    }
  }
}
```