A-to-D and D-to-A Domain Conversions (working document)

Jean-Michel Bergé, Denis Rouquier (CNET-CNS)
Serge Garcia-Sabiro (ANACAD Computer Systems)
Alain Vachoux (EPFL)

1. Introduction

This document aims to study how domain conversion could be carried on between the analog and digital parts of a VHDL-A model. It is by no mean exhaustive and is largely open to discussion.

1.1. Preliminary remarks

1) The user must be given full control over the exact transformations used in A-to-D and D-to-A interactions (from DOD document #5.1 and #6.1).

2) It is not allowed to connect *directly* a port to a pin (from DOD #6.4, i.e. without any domain conversion defined).

3) The user should not be forced to place *explicitly* extra components at the interface between A and D domains. This means that the *apparent* global connectivity of the model should not exhibit any component that will not appear in the real circuit (like the A2D and D2A components in figure 1).

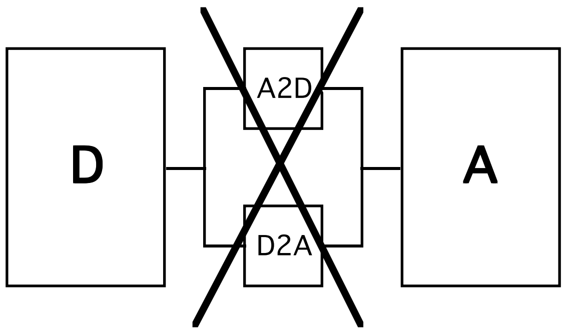

Fig. 1. Model with explicit conversion domain components.

1.2. General example

Figure 2 illustrates the general situation where a VHDL-A model consists of two parts, on digital part (D) and one analog part (A).

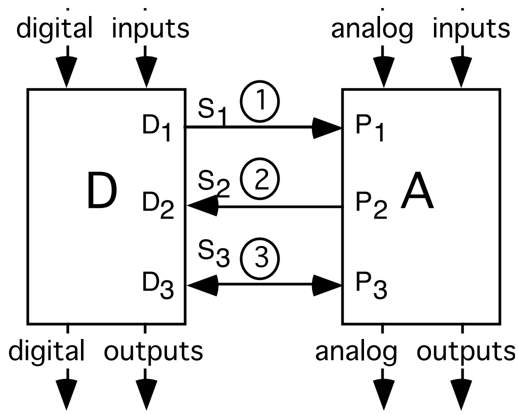

Fig. 2. General example

We can distinguish 3 kind of interactions between A and D worlds:

(1) The digital part drives the analog one, i.e. port D_1 is declared to be in mode **out**.
In that case we need a *full* D-to-A conversion: the digital values taken by the signal S_1 associated to the digital port D_1 need to be converted to an equivalent analog voltage *and* current at analog pin P_1.
On the other hand, the A-to-D conversion should only provide a model for the *fanout* of the digital port D_1. In this case, it should be an assignment delay for signal S_1.

(2) The analog part drives the digital one, i.e. port D_2 is declared to be in mode **in**.
In that case we need a *full* A-to-D conversion: the voltage at pin P_2 should be converted to an equivalent digital state and assigned to signal S_2.
On the other hand, the D-to-A conversion should only provide a model for the *fanout* of the analog pin P_2. In this case, it should be a stray capacitance.

(3) Bidirectional interaction: port D_3 is declared to be in mode **inout**. May this situation occur between digital and analog parts? Does it be explicitly forbidden?

This classification shows the need to use special *components* to handle D-to-A and A-to-D conversions. However, remark #3 of 1.1 still holds.

1.3. Full A-to-D and D-to-A conversions

We just recall here, for the sake of completeness, what should be the two *full* conversions between both domains. By *full domain conversion*, we mean explicit computation of the equivalent value of a digital signal (analog voltage and current) from an analog voltage (digital signal). On the other hand, *partial domain conversion* means the modelisation of digital (analog) load seen by the analog (digital) part.

1.3.1. Full A-to-D conversion

This conversion should use a user-supplied thresholding function posting an event (*always in the future*) on the associated digital signal only when significant changes occur in the analog part (DOD #5.2). Note that this function is highly dependent on the logic system used in the digital part (2, 3, 9, 46 states…).

1.3.2. Full D-to-A conversion

This conversion should use a user-supplied slope function (more adequate name still to be found) to avoid instantaneous change in analog voltage and current (DOD #6.3). In addition, the analog part may “see”, as a model of the digital fanin, one or several equivalent Thevenin or Norton circuits connected to the analog pin.

2. A tentative solution

We give here a proposition about the way A-to-D and D-to-A conversion may be carried out in VHDL-A. The following remarks hold:

1) Domain conversions between A and D require *two mixed components* having each a port, a pin and an architecture containing a digital process and an analog block. A more detailed description of these components will be given later.

2) These two components have a formal structure that should be defined in the language (the same way as for resolution functions for example), but may have several different behaviors. This means that several architectures of these components may exist for the same entity.

3) Syntactic short-cuts should be provided to avoid to place these components explicitly in the mixed analogue and digital VHDL model (see remark #3 of 1.1).

4) All VHDL-A examples given below do not use the final syntax.

2.1. Global domain conversion

The idea is to use the same kind of mechanism as for resolved signals. We assume first that there exists a predefined type `electrical` like in:

```
type electrical is
   record
      voltage, current: real;
      etime: time;
   end record;
```

2.1.1. Domain conversions associated to pins

The global short-cut consists of defining a subtype with associated A-to-D and D-to-A components:

```
subtype elec_conv is d2a(arch) a2d(arch) electrical;
```

`a2d(arch)` and `d2a(arch)` are "on-the-fly" configuration specifications, where the entity names `a2d` and `d2a` may be predefined in the language.

Subtype `elec_conv` is then used when declaring *pins* like in:

```
pin (…; A: elec_conv; …);
```

A pin declared this way may be directly connected to a port, because A-to-D and D-to-A conversions are associated to the pin. As soon this is the case, both `a2d` and `d2a` components are automatically invoked when domain conversion is necessary.

Remark: It should be forbidden to associate domain conversion components to *nodes*.

2.1.2. Domain conversions associated to ports

We also need domain conversion components associated to *ports*, because multiple fanin or fanout A-to-D and D-to-A connections may need to use different domain conversions (figure 3).

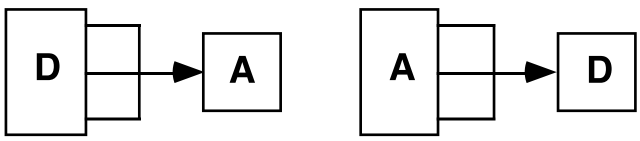

Fig. 3. Multiple fanin and fanout D/A interactions.

In that case, domain conversion components should be associated to a subtype like in:

```
subtype bit_resolved_conv is
   d2a(arch) a2d(arch) resolution_function bit;
```

Subtype `bit_resolved_conv` is then used when declaring *ports* like in:

```
port (…; D: out bit_resolved_conv; …);
```

Remarks:

1) It should be forbidden to associate domain conversion components to *signals*.

2) The subtype declaration with both domain conversion components and resolution function is quiet heavy to handle…

2.2. Local domain conversion

The global solution given in 2.1 could be locally overridden when necessary. By locally, we mean for a component declaration or a component instantiation.

2.2.1. Domain conversion associated to component declarations

For a digital component:

```
component DCOMP
   port (A: in bit; B: out bit);
   domain_conversion (A => a2d(arch_a2d1), d2a(arch_d2a1);
                      B => a2d(arch_a2d2), d2a(arch_d2a2));
end component;
```

For an analog component:

```
component ACOMP
   pin (A, B: electrical);
   domain_conversion (A => a2d(arch_a2d1), d2a(arch_d2a1);
                      B => a2d(arch_a2d2), d2a(arch_d2a2));
end component;
```

2.2.2. Domain conversion associated to component instantiation

This form should be the ultimate way to locally override the default domain conversion components.

2.3. Domain conversion components

Possible domain conversion components have already been proposed [Cott92]. We investigate here what general format they could have.

2.3.1. d2a: D-to-A domain conversion component

An effective way to carry out D-to-A conversion is to use a model called *voltage-controlled switch* (VCS) [Acun90]. Figure 4 illustrates the principle.

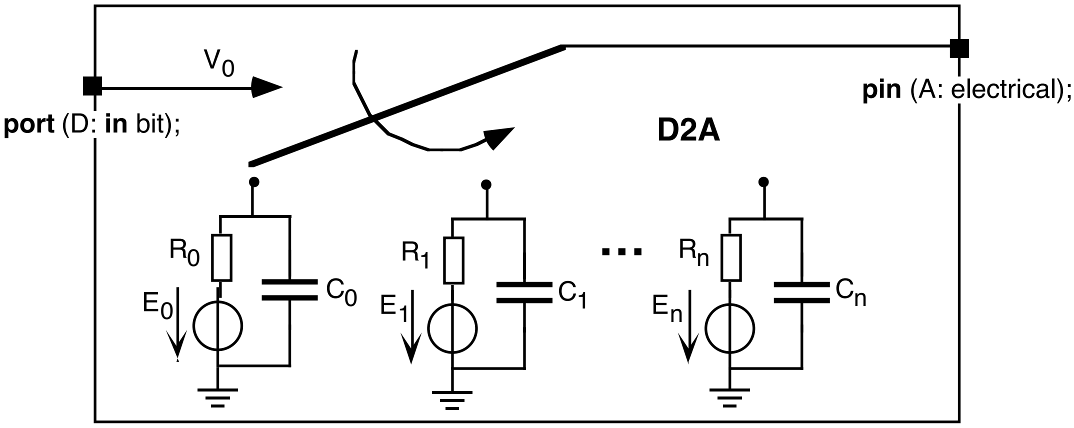

Fig. 4. Component `d2a` with voltage-controlled switch.

The component `d2a` may have the following standard entity declaration:

```
entity d2a is
   generic (R0, R1, …, Rn, C0, C1, …, Cn: real);
   port (D: in bit);
   pin  (A: electrical);
end d2a;
```

The associated architecture has still to be investigated…

2.3.2. a2d: A-to-D domain conversion component

A-to-D domain conversion may be carried out without or with hysteresis [FAS91] (see figure 5).

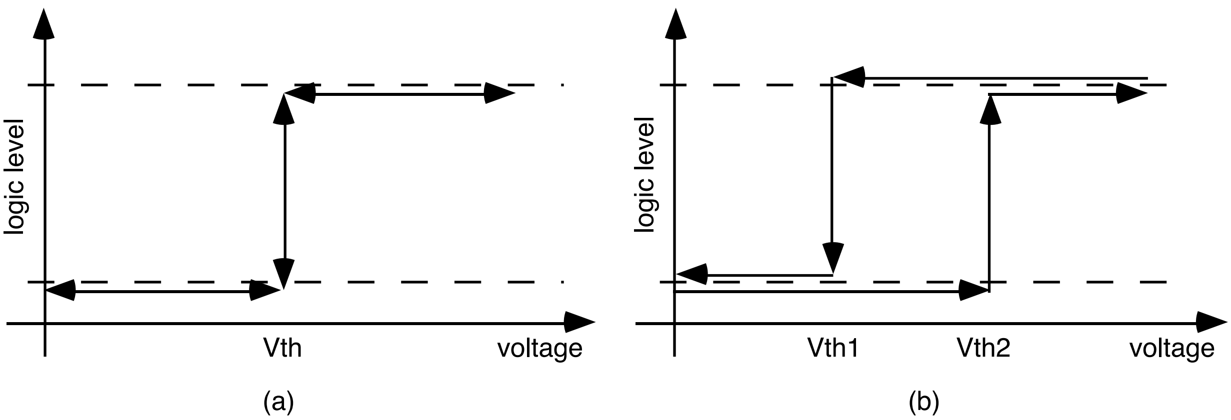

Fig. 5. A-to-D conversion (a) without and (b) with hysteresis.

The component `a2d` may have the following standard entity declaration:

```
entity a2d is
   generic (Vth, Vth1, Vth2, eps: real);
   pin  (A: electrical);
   port (D: out bit);
end a2d;
```

`eps` is a user-supplied value representing the smallest voltage change that should be noticed by the digital world.

The associated architecture has still to be investigated…

2.3.3. Where to declare the domain conversion components?

All their entity declarations may be put into the package STANDARD, while their corresponding architectures may be left to the user to implement several different domain conversion behaviors. Question arising when considering generic clauses in `d2a` and `a2d` entity specifications: Is it possible (or wise) to impose specific generic clauses that could fit every situation? If not, `d2a` and `a2d` entity specifications should not be put into package STANDARD, but also left to the user…

3. Application examples

For all examples below, we assume the following subtype declarations:

```
subtype elec_conv is d2a(arch_d2a1) a2d(arch_a2d1) electrical;
subtype bit_conv  is d2a(arch_d2a2) a2d(arch_a2d2) bit;
```

3.1. Example #1

Example #1 illustrates how the connection two components, one analog and one digital, may be described.

Let us take the circuit of figure 4.

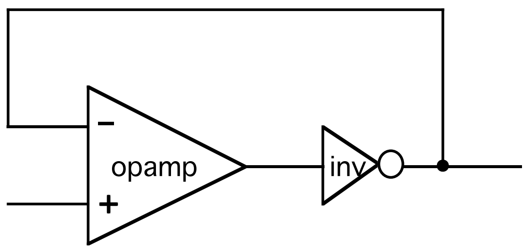

Fig. 4. Example circuit #1.

The component declarations are:

```
component INV port (IIN: in bit_conv; IOUT: out bit_conv);
component OPAMP pin (OAM, OAP, OAOUT: elec_conv);
```

One possible description of the structural model of figure 4 may then be:

```
architecture test1 of mixed_model is
   signal S1, S2: bit;
begin
   I1 : inv   port map (IIN => S1, IOUT => S2);
   OA1: opamp pin map  (OAM => S2, OAP => …, OAOUT => S1);
end test1;
```

The assignments of signals `S1` and `S2` to pins `OAM` and `OAOUT` are legal since the pins have been declared with the subtype `elec_conv`.

Another possible description is:

```
architecture test2 of mixed_model is
   node N1, N2: electrical;
begin
   I1 : inv   port map (IIN => N1, IOUT => N2);
   OA1: opamp pin map  (OAM => N2, OAP => …, OAOUT => N1);
end test2;
```

The assignments of nodes `N1` and `N2` to ports `IIN` and `IOUT` are legal since the ports have been declared with the subtype `bit_conv`.

What are the main differences between architectures `test1` and `test2`? Both involve the automatic invocation of the same domain conversion components, but architecture `test1` (`test2`) is mainly digital (analog) and nodes `N1` and `N2` (signals `S1` and `S2`) are implicit. This means the following:

1) The kind of information we can get as a result of the simulation should be logic signals for architecture `test1` and voltage/current waveforms for architecture `test2`.

2) Using nodes instead of signals when connecting analog to digital components could be useful to model transmission line effects between them (see also example #2 below).

Could it mean something else?

3.2. Example #2

Example #2 illustrates how purely digital circuits may use analog extensions to achieve a better model.

Let us take the circuit of figure 5.

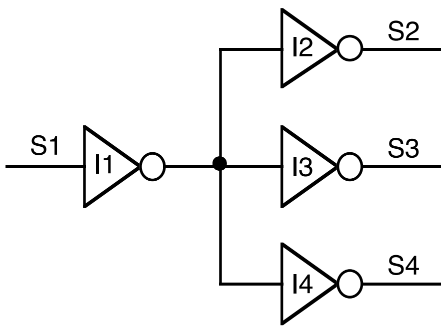

Fig. 5. Example circuit #2.

The following architecture views this circuit as purely digital:

```
architecture digital of inverters is
   signal S: bit;
begin
   I1 : inv port map (S1, S);
   I2 : inv port map (S, S2);
   I3 : inv port map (S, S3);
   I4 : inv port map (S, S4);
end digital;
```

As a consequence, all inverters `I2`, `I3` and `I4` react as soon as en event occurs at the output of `I1`.

On the other hand, the following architecture allows to take into account transmission line effects between the inverters:

```
architecture digital_with_interconnections of inverters is
   node N: electrical;
begin
   I1 : inv port map (S1, N);
   I2 : inv port map (N, S2);
   I3 : inv port map (N, S3);
   I4 : inv port map (N, S4);
end digital_with_interconnections;
```

Because node `N` is used to connect all the inverters, domain conversion components are automatically invoked when necessary. Their architectures may model specific transmission line effects, or at least the load inverter `I1` should see from its output.

4. References

[Acun90] E.L. Acuna, J.P. Dervenis, A.J. Pagones, F.L. Yang, R.A. Saleh, "Simulation Techniques for Mixed Analog/Digital Circuits", IEEE Journal of Solid-State Circuits, vol. 25, no. 2, pp. 353-363, April 1990.

[Cott92] R.A. Cottrell, VHDL Based Mixed Analog-Digital System Design, Euro-DAC '92 tutorial, Hamburg, September 1992.

[FAS91] ELDO-FAS Users's guide, ANACAD Computer Systems, January 1991.