Sampling rate and quantification density control in VHDL-AMS

Yannick Hervé, Sébastien Snaidero

CNRS-ERM/PHASE
Boulevard Sébastien Brand
67400 Illkirch-Graffenstaden
herve@erm1.u-strasbg.fr
sebastien.snaidero@ensps.u-strasbg.fr

Abstract

VHDL-AMS allows designers to describe a system with time dependent equations. If they want to plot a curve *y=f(x)*, it becomes hard to ensure the sampling rate and the quantification density of *x* and *y* values. This paper submits a very simple method to control it using a little utility model written in VHDL-AMS, called the "Q_fier".

Introduction

The simulators associated with VHDL-AMS, as for other languages, allow the designers to control only a few parameters to insure the correctness of the simulation. There are principally the maximum and minimum time steps, the integration algorithm, and the accuracy (or error). Algorithms implemented in simulators have to find the best time step between the limit steps according to a convergence criterion.

The "accuracy" parameter which is often accessible through a single value is composed of a relative value and an absolute value. This setting allows the simulator to define whether an ASP (Analog Simulation Point) is correct or not. As it is hard to know how the two components are computed, the user is not given to use this parameter to force a good quantification.

When the model is composed of a set of temporal equations it is relatively easy to choose suitable values of time steps that ensure a bounded variation of the quantities amplitudes between each ASP. However it becomes quite impossible to choose the right settings that ensure a sufficient quality of quantification when the interesting result is, for instance :

$$\boldsymbol{x}(t) = f(\boldsymbol{y}(t)) \qquad (1)$$

As an example, we wished to model a *B(H)* magnetic curve. We knew the $\boldsymbol{H}(t)$ signal and we designed a model that computed the hysteretic non-linear $\boldsymbol{B}(t)$ signal with a Preisach approach. Then we had to face the problem: how to choose the right time step that would allow to mesh properly the B(H) plan? It becomes even worst when the same model is used in a complete system: the global voltage $\boldsymbol{u}(t)$ is set and the simulator computes the current $\boldsymbol{i}(t)$ through $\boldsymbol{H}$(t) and $\boldsymbol{B}$(h) which are calculated according to the derivative of $\boldsymbol{u}$(t).

To control the quality of the amplitude "quantification", it is possible to impose a temporal step according to the maximum slope that can be admitted for the quantities in the model. In many cases this value is difficult to get. In the approach presented below, the solution is to turn the variable time step algorithm into a pseudo time-driven simulator. This piece of model can be inserted anywhere in a testbench, with "time_driven" a boolean signal:

```
time_driven   <= not time_driven
                 after time_step;
break on time_driven;
```

These lines forces the simulator to compute an analog point each time there is an event on the "time_driven" signal. However, this device is only a maximum boundary for the time variations between two consecutive edges of the "time_driven" signal and the simulator settings can therefor lead to the insertion of additional ASP. In such a case, the user has no control on the minimum variations of the simulated quantities between two analog simulation points, but increasing the minimum time step of the simulator.

Presentation of the "Q_fier"

The method we submit is based on a VHDL-AMS model, the "Q_fier" (quantifier), that gives a simple solution to this problem. This model can be instanciated anywhere in the testbench to with the following interface:

```
uut : entity Q_fier(beh)
      generic map (time_step,ampl_step)
      port map (constrained_quantity);
```

This sub model is built not to interfere with the one in which it is instanciated. It just forces the simulator to compute an analog simulation points considering the evolution of the "*constrained_quantity*" quantity according to the "*time_step*" and "*ampl_step*" that bounds the time elapsed and the amplitude variation since the last time and amplitude reference point. If the simulator computes time steps or value variations lesser than constraints, the tool is asleep (Fig. 1).

It is possible to instanciate more than one "Q_fier" to constraint several quantities. Note that the computation architectures of actual simulator, based on a single sampling time for all quantities, let the more constrained Q_fier drives the simulation rate bound.

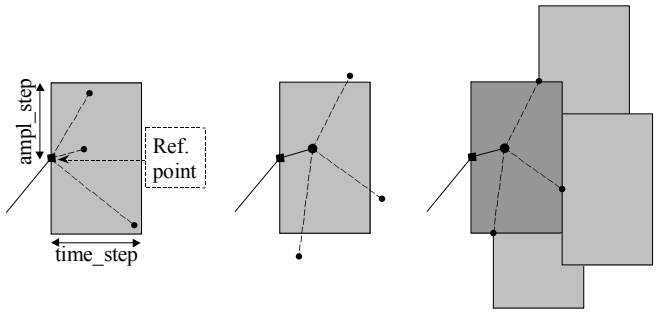


Fig. 1.A. The ASPs submitted by the simulator (end of dash lines) are accepted as they are in the boxe.

Fig. 1.B. A first ASP has been validated and the following ones (end of dash lines) are out of the box. Then they are rejected and the "Q_fier" imposes new ASPs (ie. Fig. c).

Fig. 1.C. The points imposed by the "Q_fier" match the border of the box and are the new references points for the next "Q_fier" boxes.

Fig. 1. "Q_fier" principle

Code explanation

According to the instanciation explanations above, the interface of the following model works on a specified *quantity* and realize the sampling according to the T_step and Q_step parameters placed as generic data for reusability:

```
entity Q_fier is
   generic    (T_step:real:=1.0e-2;
               Q_step:real:=1.0e-2);
   port       (quantity Q: in real);
end entity Q_fier;
```

To monitor the specified quantity, the architecture of the "Q_fier" manages two quantities (*Q_diff* and *T_diff*) that represent the evolution of its two components: time and amplitude for the last reference point. When those differences exceed the set up steps, the use of the ***'above*** instruction theoretically creates a break point at the cross time.

```
architecture beh1 of Q_fier is
   quantity   Q_diff, T_diff : real;
   signal     T_hold : real := 0.0;
   signal     Q_hold : real := Q;
   signal     V_spy : boolean;

   begin
      break on V_spy;

      Q_diff == abs(Q - Q_hold);
      T_diff == now - T_hold;
      V_spy  <= Q_diff'above(Q_step) or
                          T_diff'above(T_step);
```

When the break point is reached, an internal signal unlatch a process that sets new values for the amplitude and time references:

```
   New_Memory : process begin
      wait until V_spy;
      Q_hold <= Q;
      T_hold <= now;
   end process;
end architecture;
```

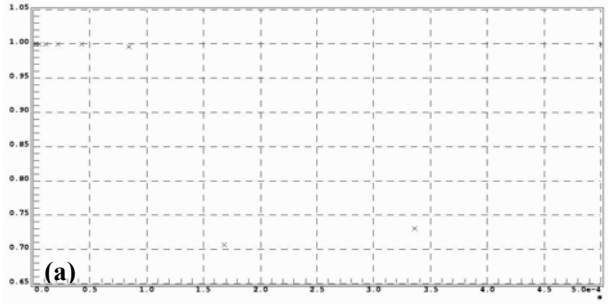
Examples

To illustrate our purpose about the "Q_fier", we will use a quite simple example based on a set of three connected functions:

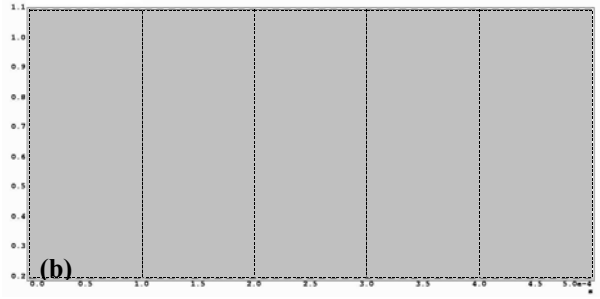
$$x(t) == \alpha.\sin(2.\pi.\beta.t)$$
$$y(t) == x(t)\text{^}4$$
$$z(t) == \gamma.\cos(y(t))\text{^}2$$

All the simulations that follow were lead with the Mentor Graphics software ***ADVance MS 1.4_1.2***.

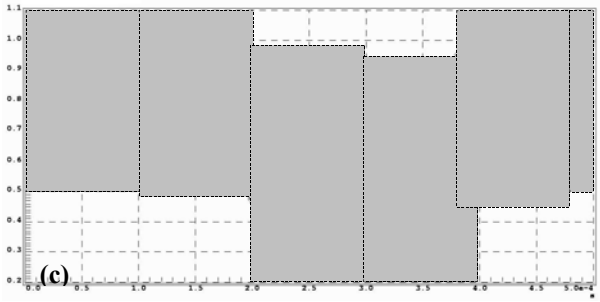
First, to illustrate the influence of the "Q_fier" on a temporal function, we have extracted some graphics corresponding to different settings of the "Q_fier" placed on the *z(t)* quantity (cf. Fig. 2). For those tests, we realized a simulation of 5^{E}-4 s with the minimum and maximum steps set respectively at 1^{E}-12 and 1^{E}-3. For those test, we kept the *T_step* parameter of the "Q_fier" set on 1^{E}-4. Note that we have shadowed the areas corresponding to the "Q_fier" boxes on the following graphics. It means that points not located on the borders of the boxes are not the fact of the "Q fier". (α=1.0, β=1000.0, γ=1.0):



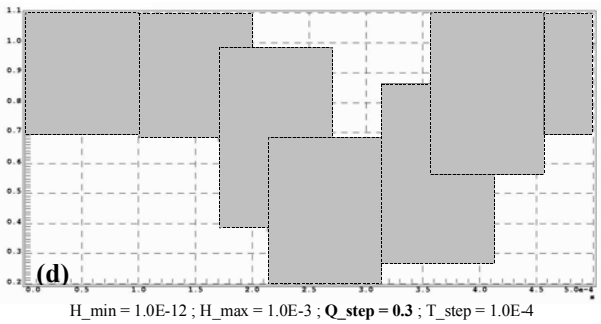
H_min = 1.0E-12 ; H_max = 1.0E-3 ; without "Q_fier"



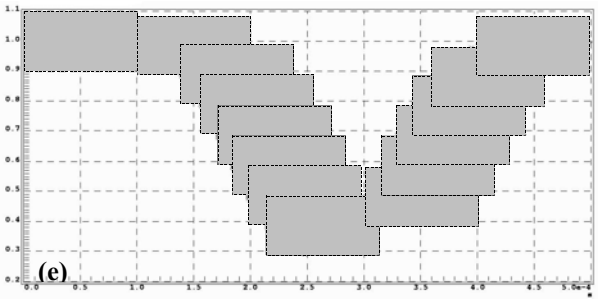
H_min = 1.0E-12 ; H_max = 1.0E-3 ; **Q_step = 1.0** ; T_step = 1.0E-4



H_min = 1.0E-12 ; H_max = 1.0E-3 ; **Q_step = 0.5** ; T_step = 1.0E-4



H_min = 1.0E-12 ; H_max = 1.0E-3 ; **Q_step = 0.3** ; T_step = 1.0E-4

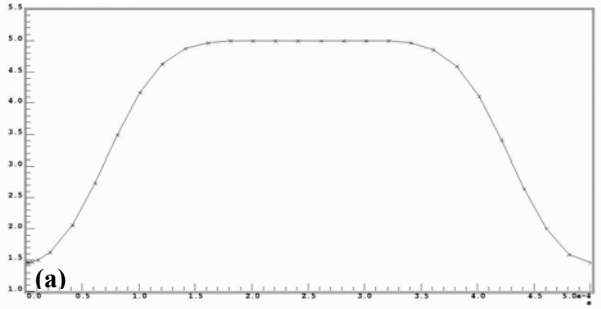


H_min = 1.0E-12 ; H_max = 1.0E-3 ; **Q_step = 0.1** ; T_step = 1.0E-4

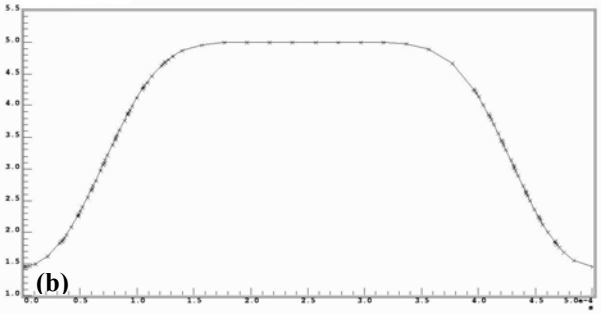
Fig. 2. *z(t)* sampling and quantification evolution with a "Q_fier"

Fig. 2 does well show that it is possible to control the sampling rate of the simulator taking into account the evolution of the amplitude of the signal.

If the time constraint is correctly relaxed, the "Q_fier" will not create any point where it is not required, as in Fig. 3. For those tests, the simulator settings are exactly the same as before, except the maximum time step which is 1^{E}-4. The *z(t)* function has been dilated for a better view (γ=5.0), and it has been phased out for better representation:



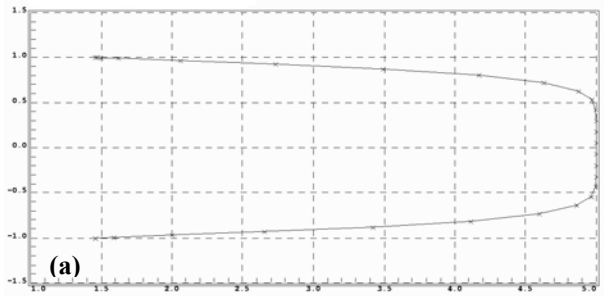
H_min = 1.0E-12 ; H_max = 1.0E-4 ; without "Q_fier"



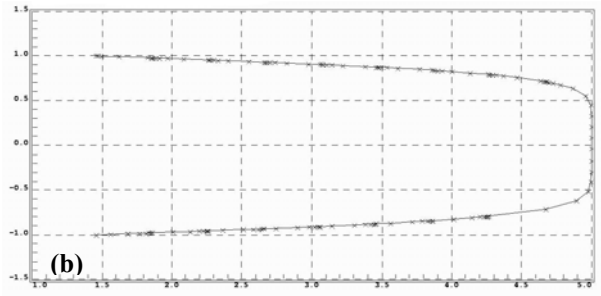
H_min = 1.0E-12 ; H_max = 1.0E-4 ; **Q_step = 0.4** ; T_step = 1.0E-4

Fig. 3. *z(t)* selective action demonstration

The final example deals with the ***z=f(x)*** representation. The following graphics were obtained with the same settings as previously. It shows how the "Q_fier" dynamically mange the sampling to get a minimum resolution:



H_min = 1.0E-12 ; H_max = 2.0E-5 ; without "Q_fier"



H_min = 1.0E-12 ; H_max = 2.0E-5 ; **Q_step = 0.4** ; T_step = 1.0E-4

Fig. 4. ***z=f(x)*** representations

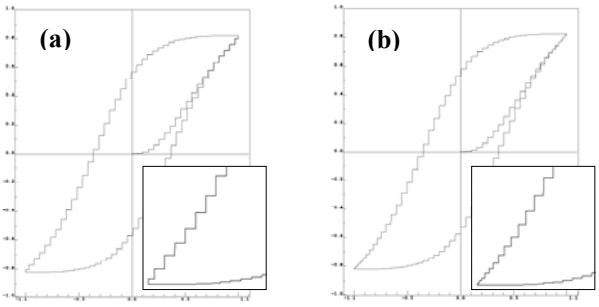
The representation is definitely better on Fig. 4.b. That could allow to locate some anomaly in models that may not be detected in other ways.

A scheduler

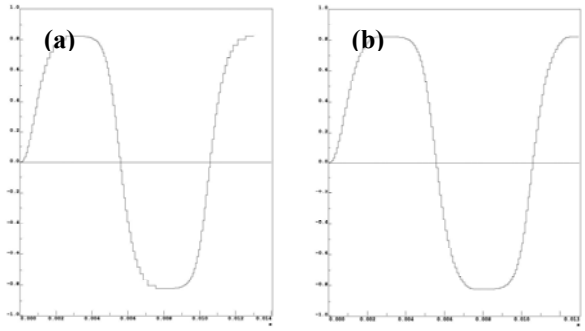
Another use of this model is to make up for the PROCEDURAL instruction which is not available on current simulators.

A first idea to replace this missing instruction could be to use a PROCESS scheduled on each ASP. Unfortunately, there is no signal returned by the simulators which notifies an ASP has been computed. Such a signal can not be returned in the current standard due to portability difficulties.

A solution to this problem is to use a "Q_fier". Indeed when you want to compute ***y=f(x)*** with some complex dependencies, you would like too associate a value to ***y*** for each ASP on ***x***. With the "Q_fier", this can be done at each time the update process is engage. In this way, the ***y*** quantity remains constant between two of those steps. In the hysteretic problem mentioned in the introduction, the nonlinear relations were responsible of fluctuating precision on ***B(t)***. As shown bellow, the use of the "Q_fier" helped to get more regular sampling and better representation of ***B(H)***.



Hysteresis cycle computation without *(a)* and with *(b)* a "Q_fier" (***B(H)***)



Magnetization computation without *(a)* and with *(b)* a "Q_fier" (***B(H)***)

Fig. 5. Illustration of the "Q_fier" as a scheduler

Conclusion

References

[1] Y. Hervé, "VHDL-AMS : Applications et enjeux industriels" Dunod éditeur, 2002.

[2] 1076.1-1999 IEEE Standard VHDL Analog and Mixed-siggnal Extensions (Language Reference Manual [LRM])

[3] E. Christen et al, "Tutorial VHDL-AMS", 36e, Design Automation Conference, New Orlean, june 21-25, 1999.