16

Behavioural Modelling of Sampled-Data Systems with HDL-A and ABSynth

Vincent Moser, Hans Peter Amann, Fausto Pellandini
Institute of Microtechnology, University of Neuchâtel
Rue A.-L. Breguet 2, CH-2000 Neuchâtel, Switzerland
Tel. +41 32 718 3414, fax +41 32 718 3402
E-mail vincent.moser@imt.unine.ch

Abstract

In this paper we make use of a mixed analogue-digital HDL to model sampled-data systems. Modelling solutions are presented to code some sampled-data behaviour using either analogue or digital constructs of the language.

Two key operations – sampling and shifting – are presented first. Then, these facilities are used to code difference equations. This approach is validated on the example of a sampled-data filter. Finally, a 'real-life' switched-capacitor A to D converter is modelled and simulated. The results are compared with a transistor-level description.

Keywords

Behavioural modelling, analogue hardware description language, automatic code generation, sampled-data systems.

1 INTRODUCTION

Usually, mixed-mode hardware description languages (HDLs) are used to describe analogue, digital or mixed analogue-digital systems. In this paper, we will concentrate on the behavioural modelling of a particular class of systems called sampled-data systems. They process signals that have a continuous amplitude but that are only defined at particular points in time. Such signals can be described as series of real

numbers called samples. A typical class of sampled-data systems is formed of switched-capacitor circuits.

As sampled-data systems constitute an intermediate class of systems between analogue and digital systems, we will describe them using either analogue HDL descriptions or digital HDL constructs. Accordingly, they will be simulated either with analogue simulation algorithms or with an event-driven simulator. Simulation results and simulation time will be used to compare the two coding approaches. Finally, sampled-data systems will be modelled with the automatic model generator ABSynth.

As the IEEE 1076.1 VHDL-AMS is not on the market yet (Dec. 1996), we use the language HDL-A.

In section 2 a brief overview of HDL-A and of ABSynth will be given, then in section 3 we will expose various modelling solutions. Section 4 will be devoted to the modelling of an A to D converter as an application example. Finally, some conclusions will be drawn.

2 HDL-A AND ABSYNTH IN SHORT

HDL-A (ANACAD, 1994) is a purely behavioural language by ANACAD; structural description must be given in the form of Spice-like netlists. An HDL-A model – like a VHDL model – is composed of two parts: an 'entity' declaration and an 'architecture' body.

The entity describes the interface of the model including analogue pins, digital ports and static parameters. The architecture body describes the behaviour of the system. It includes a 'relation' block for analogue descriptions and a 'process' block for digital descriptions. The process block is a sub-set of the behavioural modelling facility of VHDL. The analogue part can contain procedural statements as well as differential equations.

Several objects can be used to describe behaviour in HDL-A: a 'state' represents an analogue quantity, a 'signal' represent a digital waveform, while neutral objects 'constant' and 'variable' also exist.

ABSynth (Analogue Behavioural model Synthesizer) (Moser et al., 1995) is a computer-aided model generator. The behaviour of the model is described in the form of a functional diagram (FD) – an extended block diagram – drawn as the interconnection of graphical building symbols (GBSs). Each GBS stands for an analogue functionality. A library of standard GBSs exists which includes basic mathematical operators, function generators and particular interface elements. Each GBS is associated to a code template which is used by ABSynth to generate a complete, purely analogue, HDL-A model. The GBS library can be extended with new GBSs either using hierarchy or in

association with new code templates. In this paper, we will develop 'sampled-data' GBSs and use them to model sampled-data examples with ABSynth.

3 SAMPLED-DATA MODELLING

In this section we expose the various modelling styles that can be applied to model sampled-data systems in HDL-A. For this purpose, we first need to model two key operations: the sampling of a signal and the shifting of a signal by a given number of clock cycles. Then we will see how to model a system given by a difference equation or by a z-domain transfer function. More information on the z-transform and on the description of digital systems can be found in (Kuc, 1988).

3.1 Sampling and Shifting Operations

Let us see now how to implement two operations that are key issues in sampled-data modelling: the sampling operation and the shifting operation. The sampling operation is important in signal processing to convert an analogue input signal into a series of samples. The shifting operation is important each time a signal is computed as a function of past values of some signals. Some solutions will be developed to code them either in the digital or in the analogue domain. In any case, we assume that the input signal $x(t)$ is band-limited according to the sampling theorem.

Coding in the Digital Part

In the digital part of an HDL-A model, like in VHDL, the information is carried by objects called 'signals'. A signal is defined at discrete points on the time axis. Its amplitude can either be quantified or not. A signal is computed according to an algorithm coded in a 'process' block and controlled by a 'wait' statement. A signal takes its actual value after the wait statement and the corresponding algorithm have been completely executed. This value will then be held until the next execution of the wait statement.

To model sampled-data systems, we make the following choices:

- In order for the signals to match our definition of sampled-data signals, they will not be quantified.
- The wait statement will depend only on the clock and not on the results of any other process.

We are then sure that the whole system is open-loop and that each wait statement will be executed only once per clock cycle. As a corollary, if we read an analogue state variable inside the wait loop, it will be read only once per cycle, hence realizing the sampling operation. Furthermore, the value of the previous sample of each signal is

available during the computation of the algorithm. This way, a shifted copy of a signal can be obtained just by assigning its value to another signal.

The following code example illustrates the sampling of the input state variable $x(t)$ into the signal $x_1(k)$ and the shifting of $x_1(k)$.

```
PROCESS
...
LOOP
   WAIT ON ...;
      x1 <= x;     -- sampling of the state x into the signal x1
      x2 <= x1;    -- shifting of the signal x1 into the signal x2
      ...
END LOOP;
END PROCESS;
```

Coding in the Analogue Part

Alternatively, the same functions can be realized in the analogue part of a model. According to the principles of analogue simulation, any equation or piece of procedural code is considered as being always satisfied and can be evaluated at any time depending on the way the simulator manages the time. Consequently, there are no trivial sampling and shifting mechanisms in the analogue part of HDL-A. We will have to implement both of them.

Basically, the sampling and shifting operations must be synchronized to the clock signal. A conditional statement controls the execution of a particular block in which the analogue input signal will be sampled and some sampled signals will be shifted by an integer number of clock periods. For example, the condition can be set to the rising edge of the clock signal.

```
IF rising(vclk,threshold) THEN
```

The sampling operation is simply carried out as an assignment statement. The instantaneous value of the continuous input variable $x(t)$ is copied into another state variable $x_0(t)$ at sampling time.

```
x0 := x;
```

The sampling period is then calculated as the difference between the current time and the last sampling time.

```
tdel := real(current_time - previous(tmem));
tmem := current_time;
```

Then, some sampled signals can be shifted by one or more sampling periods. This is done using the function 'previous' of HDL-A. The function previous(x) returns the value of a state x at the last computation instant, while the function previous(x,t) returns the – eventually interpolated – value of x a certain amount of time t earlier.

```
x1 := previous(x0,(tdel));
```

These sampling and shifting mechanisms can now be used to model sampled-data behaviour specified, e.g., by difference equations.

3.2 Coding of a Difference Equation

If the sampling is periodic in time and synchronous for all signals, the system can be described by a difference equation or by a z-domain transfer function. We will first see how to code a difference equation in the digital part of the model code, then in the analogue part. Second, we will derive the difference equations that correspond to elementary z-domain cells. We will limit the discussion to linear systems described by difference equations with constant coefficients of the form

$$y(k) = \sum_{m=0}^{M} a_m x(k-m) - \sum_{n=1}^{N} b_n y(k-n) \qquad (1)$$

Coding in the Digital Part

We evaluate the signal $y(k)$ as a function of the input state $x(t)$ and of the shifted signals $x_m(k)$ and $y_n(k)$ and according to the sampling and shifting facilities defined above. We get the following pseudo-code.

```
PROCESS ...
LOOP
   WAIT ON rising(vclk,threshold);
      x1 <= x;                              -- sampling of x into x1
      x2 <= x1;                             -- shifting of x1 into x2
      ...
      xm <= x[m-1];
      y <= a0*x + a1*x1 + ... + am*xm - b1*y - ... - bn*yn; -- diff eq
      y2 <= y;                              -- shifting of y into y2
      y3 <= y2;                             -- shifting of y2 into y3
      ...
      yn <= y[n-1];
END LOOP;
END PROCESS;
```

Note that the wait statement could also be expressed in the form:

```
WAIT UNTIL rising(vclk,threshold);
```

Coding in the Analogue Part

Here, we evaluate the response $y(t)$, which is now an analogue state variable as a function of the shifted states $x_m(t)$ and $y_n(t)$. The analogue sampling and shifting operations are implemented as explained above. We get the following pseudo-code.

```
PROCEDURAL FOR transient =>
...
IF rising(vclk,threshold) THEN
   x0 := x;                              -- sampling of x into x0
   tmem := current_time;                 -- storage of the current time
   tdel := real(current_time - previous(tmem));      -- clock period
   x1 := previous(x0,tdel);          -- shifting of x0 by 1 clock cycle
   ...
   xm := previous(x0,(m*tdel));      -- shifting of x0 by m clock cycles
   y1 := previous(y,tdel);           -- shifting of y by 1 clock cycle
   ...
   yn := previous(y,(n*tdel));       -- shifting of y by n clock cycles
END IF;
y := a0*x0 + a1*x1 + ... + am*xm - b1*y1 - ... - bn*yn;   -- diff eq
```

3.3 Z-Domain Elementary Cells

We have seen above two solutions to code a difference equation in HDL-A. The same specification can also be expressed as a rational z-transfer function which can be decomposed as a product of elementary cells. To obtain the same behaviour, the corresponding elementary difference equations are coded separately and cascaded.

The z-transform of (1) gives

$$H(z)=\frac{Y(z)}{X(z)}=\frac{\sum_{m=0}^{M} a_m z^{-m}}{1+\sum_{n=1}^{N} b_n z^{-n}}, \qquad (2)$$

which can be developed as a quotient of two products. The terms of the numerator represent the zeros of the transfer function; the terms of the denominator represent the poles of the transfer function. We get:

$$H(z) = H_0 \frac{\prod_{m=1}^{M}(1-\zeta_m z^{-1})}{\prod_{n=1}^{N}(1-p_n z^{-1})}. \tag{3}$$

We now code these elementary pole-zero cells in HDL-A in both the digital and the analogue versions.

Real Pole Combined with a Zero on the Origin

The single pole is located at coordinate p on the real axis. The transfer function – normalized for $H(z=1) = 1$ – gives:

$$H(z) = (1-p)\frac{1}{1-pz^{-1}}. \tag{4}$$

The corresponding difference equation can be coded in digital HDL-A as

```
y <= (1.0-p)*x + p*y
```

and in analogue HDL-A as

```
y := (1.0-p)*x0 + p*y1
```

Pair of Complex Conjugate Poles Combined with a Double Zero on the Origin

The poles are located at coordinates $p=\alpha_p+j\beta_p$ and $p^*=\alpha_p-j\beta_p$. The transfer function gives:

$$H(z) = (1-p)(1-p^*)\frac{1}{(1-pz^{-1})(1-p^*z^{-1})} \tag{5}$$

The corresponding difference equation can be coded in digital HDL-A as

```
y <= (1.0-2.0*ap+ap**2+bp**2)*x + 2.0*ap*y - (ap**2+bp**2)*y2;
```

and in analogue HDL-A as

```
y := (1.0-2.0*ap+ap**2+bp**2)*x0 + 2.0*ap*y1 - (ap**2+bp**2)*y2;
```

Zero Combined with a Pole on the Origin

The single zero is located at coordinate ζ on the real axis. The transfer function gives:

$$H(z) = \frac{1}{(1-\zeta)}\left(1-\zeta z^{-1}\right) \tag{6}$$

The corresponding difference equation can be coded in digital HDL-A as

```
y <= (x-z*x1)/(1.0-z)
```

and in analogue HDL-A as

```
y := (x0-z*x1)/(1.0-z)
```

Pair of Complex Conjugate Zeros Combined with a Double Pole on the Origin.

The zeros are located at coordinates $\zeta=\alpha_z+j\beta_z$ and $\zeta^*=\alpha_z-j\beta_z$. The transfer function gives:

$$H(z) = \frac{1}{(1-\zeta)(1-\zeta^*)}\left(1-\zeta z^{-1}\right)\left(1-\zeta^* z^{-1}\right) \tag{7}$$

The corresponding difference equation can be coded in digital HDL-A as

```
y <= (x-2.0*az*x1+(az**2+bz**2)*x2)/(1.0-2.0*az+az**2+bz**2);
```

and in analogue HDL-A as

```
y := (x0-2.0*az*x1+(az**2+bz**2)*x2)/(1.0-2.0*az+az**2+bz**2);
```

If we connect these cells in cascade to build an arbitrary transfer function, the exceeding singularities we introduced at z=0 cancel each other. These elementary z-domain functionalities have been implemented as ABSynth GBSs and can consequently be used to model z-domain transfer functions with ABSynth.

3.4 Example: Chebychev Low-Pass Filter

As a first example, a third-order Chebychev low pass-filter with a cut frequency of 0.1 Hz is modelled (Kunt, 1984). The z-domain transfer function is given by

$$G(z) = \frac{0.0154 + 0.0461z^{-1} + 0.0461z^{-2} + 0.0154z^{-3}}{1 - 1.9903z^{-1} + 1.5717z^{-2} - 0.458z^{-3}} \tag{8}$$

This transfer function will be coded in several different ways which will be compared in terms of resulting curve shape and simulation time. The input pin and the clock pin are both modelled as an RC input stage, the output pin is modelled as an output resistance as explained in (Moser et al., 1994).

Coding of the difference equation

By applying the inverse z-transform, we get the difference equation

$$\begin{aligned} y(k) = {} & 0.0154x(k) + 0.0461x(k-1) + 0.0461x(k-2) + 0.0154x(k-3) + \\ & 1.9903y(k-1) - 1.5717y(k-2) + 0.458y(k-3) \end{aligned}, \tag{9}$$

which can be coded in digital HDL-A as

```
yout <= 0.0154*vin + 0.0461*yin + 0.0461*y2in + 0.0154*y3in +
        1.9903*yout + 1.5717*y2out +0.458*y3out
```

or in analogue HDL-A as

```
yout := 0.0154*samin + 0.0461*tyin + 0.0461*t2yin + 0.0154*t3yin
      + 1.9903*tyout + 1.5717*t2yout + 0.458*t3yout
```

Pole-zero Decomposition, ABSynth Modelling

The transfer function can also be decomposed into a product of elementary cells as described above. We get

$$G(z) = \frac{\left(1 + z^{-1}\right) \cdot \left(1 + 1.993506z^{-1} + z^{-2}\right)}{\left(1 - 0.66062z^{-1}\right) \cdot \left(1 - 1.32968z^{-1} + 0.693287z^{-2}\right)} \tag{10}$$

The singularities of this transfer function are then one single zero, one single pole, one pair of complex conjugate zeros and one pair of complex conjugate poles. This version of the model was coded using ABSynth according to the FD of figure 1. Each z-domain singularity is represented by a GBS.

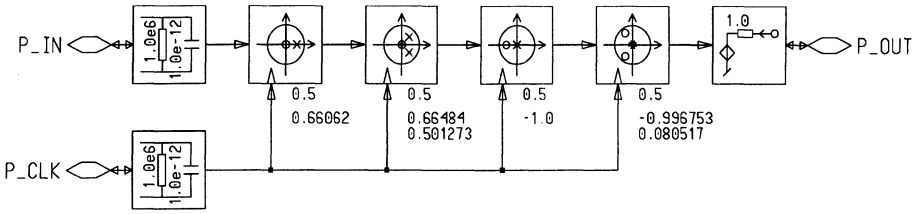

Figure 1 Functional diagram of a sampled-data filter

Simulation results

The various models were simulated in transient mode with a sample frequency of 1 Hz. Figure 2 shows the resulting curves for an input unity step. The response of an equivalent analogue model is given for comparison purposes. We see that the results of the different model variants are very similar. Note: the 'digital' model is actually a mixed-mode model because its interface is analogue.

The CPU time necessary for this simulation on a Sun Sparc 10/30 is nearly identical for the digital model and for the analogue model. The ABSynth model, however, is slower by a factor of 2. This is due to the fact that the HDL-A code generated by ABSynth is not optimized and it contains a larger number of state variables (Moser, 1996).

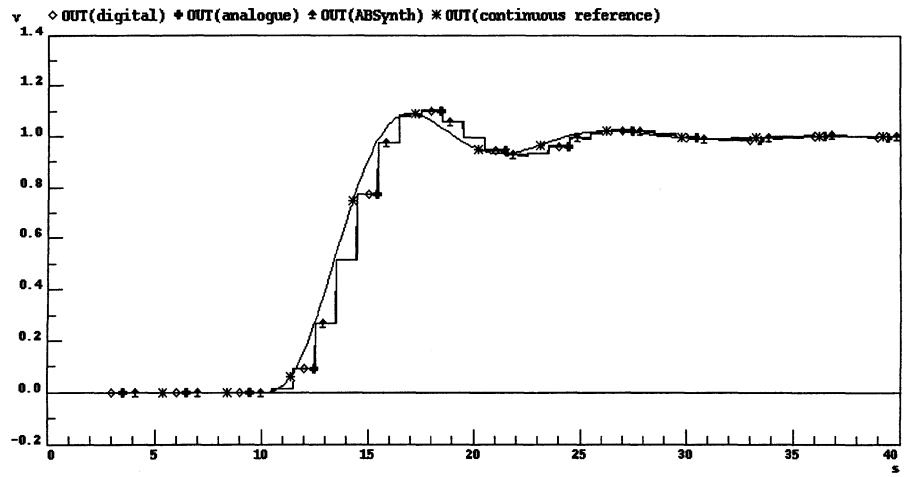

Figure 2 Transient simulation of the sampled-data filter

4 APPLICATION: RSD A/D CONVERTER

As a more complete application example, a cyclic RSD (redundant signed digit) analogue to digital converter is modelled. Different variants of the converter model will be coded and compared with a switched-capacitor CMOS implementation in terms of accuracy and simulation time.

4.1 Principle of Operation

The Redundant Signed Digit (RSD) converter (Ginetti et al., 1992) performs a successive approximation conversion based on a two-level comparison and produces a ternary output signal V_{out} according to the RSD algorithm represented in figure 3.

The input signal V_{in} is compared with two threshold voltages $+V_{th}$ and $-V_{th}$. According to the result of the comparison, the first bit takes the value 1, 0 or –1. The input signal is then multiplied by 2 and a reference voltage V_r is either added or subtracted to obtain a new intermediate signal V_x. The same comparison algorithm is then applied recursively to V_x in order to calculate all the n output bits.

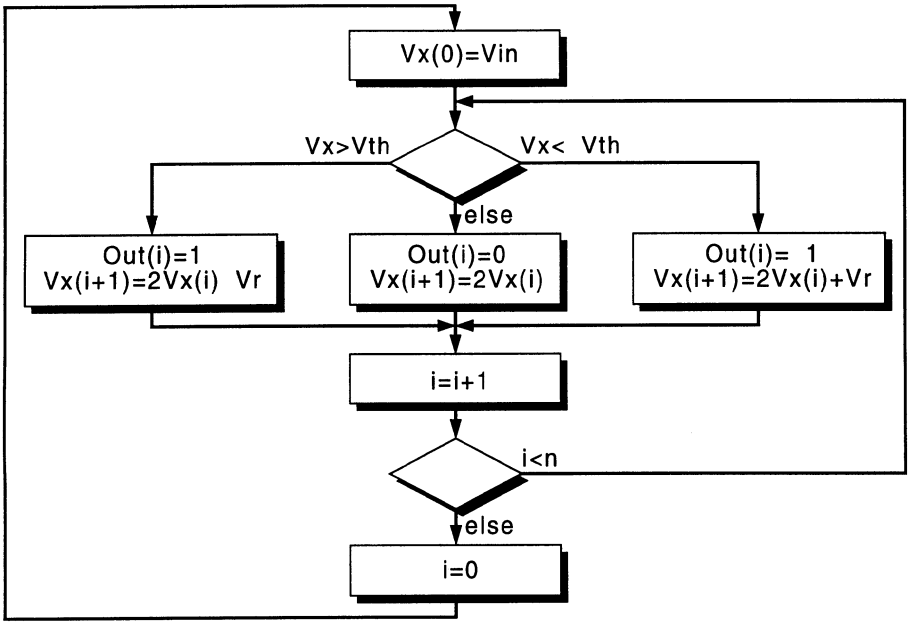

Figure 3 Cyclic RSD conversion algorithm

4.2 Behavioural Modelling

The RSD converter is now modelled in HDL-A. The ternary result is expressed using two binary signals: the positive one, outp, is set to high when vx is 1 and to low otherwise, the negative one, outn, is set to high when vx is –1 and to low otherwise. Three model variants are proposed. Firstly, the core algorithm is expressed as analogue code in a 'relation' block. Secondly the algorithm is coded using digital constructs in a 'process' block. Finally, the converter is modelled with ABSynth, which gives another analogue model. In all cases we resort to the sampling and shifting facilities presented above.

Besides the core of the algorithm, a complete analogue interface (Moser et al., 1994) is also modelled as well as some secondary effects like output delay and inactive output voltage.

Digital Core

The computation is triggered off by the rising edge of the digital clock signal.

```
WAIT ON clk UNTIL event(clk) AND clk='1';
```

Then, the variable vxin is loaded either with a sample of the input vin or with vx. Another signal count is used to count the n bits.

```
IF (count=n-1.0) THEN
   count <= 0.0;                    -- initialize bit counter
   vxin := vx;                      -- laod vx for lsb
ELSIF (count=0.0) THEN
   count <= 1.0;                    -- increment bit counter
   vxin := vin;                     -- sample input vin
ELSE
   count <= count + 1.0;            -- increment bit counter
   vxin := vx;                      -- laod vx
END IF;
```

Finally, vx is computed according to the actual algorithm and shifted; the two digital outputs outp and outn are set to 1 or 0.

```
IF (vxin>v_thr) THEN                -- upper comparison
   vx <= 2.0*vxin - v_ref;          -- new vx value computation
   outp <= '1';                     -- +1 output
   outn <= '0';
ELSIF (vxin<-v_thr) THEN            -- lower comparison
```

```
      vx <= 2.0*vxin + v_ref;          -- new vx value computation
      outp <= '0';
      outn <= '1';                     -- -1 output
   ELSE
      vx <= 2.0*vxin;                  -- new vx value computation
      outp <= '0';                     -- 0 output
      outn <= '0';
   END IF;
END LOOP;
...
END PROCESS;
```

Analogue Core

The time representation used in this analogue description is continuous but the algorithm to implement – the behaviour of the model – is typically of sampled-time type. Therefore, we trigger off the execution of the corresponding code block on the rising edge of the clock.

```
IF RISING(vclk,clk_thr) THEN
```

First, the sampling period `tdel` is computed as the difference between the current time and the last sampling time stored in `tmem`.

```
   tmem := current_time;
   tdel := real(current_time - previous(tmem));
```

Then, the state variable `vxin` is loaded either with the value of the input `vin` – to compute a new input sample – or with the last computed value of `vx`.

```
   IF (previous(count)=n-1.0) THEN
      count := 0.0;                    -- initialize bit counter
      vxin := previous(vx,tdel);       -- laod vx for lsb
   ELSIF (previous(count)=0.0) THEN
      count := 1.0;                    -- increment bit counter
      vxin := vin;                     -- sample input vin
   ELSE
      count := previous(count) + 1.0;  -- increment bit counter
      vxin := previous(vx,tdel);       -- laod vx
   END IF;
```

Finally, `vx` is calculated according to the actual algorithm; the two outputs `outp` and `outn` are set to VDD to code a 1 and to VSS to code a 0.

```
    IF (vxin>v_thr) THEN                    -- upper comparison
         vx := 2.0*vxin - v_ref;            -- new vx value computation
         voutp := vvdd;                     -- +1 output
         voutn := vvss;
     ELSIF (vxin<-v_thr) THEN               -- lower comparison
         vx := 2.0*vxin + v_ref;            -- new vx value computation
         voutp := vvss;
         voutn := vvdd;                     -- -1 output
    ELSE
       vx := 2.0*vxin;                      -- new vx value computation
       voutp := vvss;                       -- 0 output
       voutn := vvss;
    END IF;
END IF;
```

The reference voltage `v_ref`, the threshold voltage `v_thr`, and the number of conversion cycles `n` were some of the static parameters of the models.

ABSynth Modelling

The RSD converter has also been modelled using ABSynth.

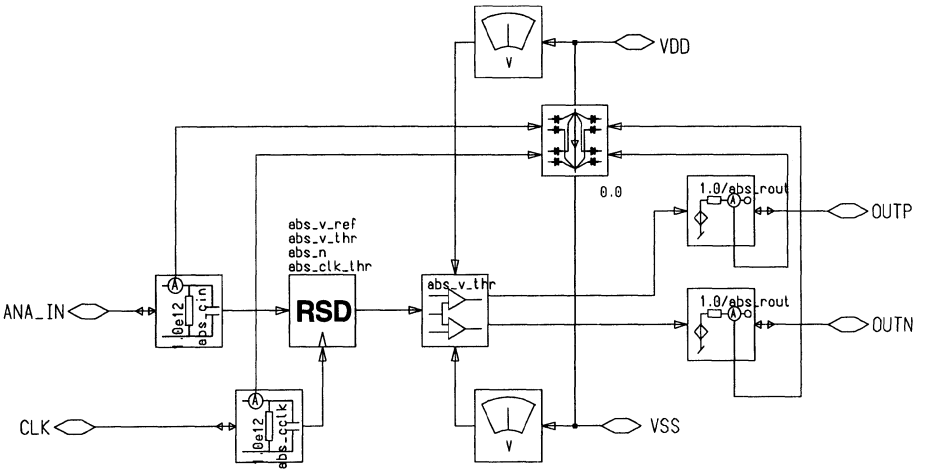

Figure 4 Functional diagram of the RSD A to D converter

The behaviour considered here is exactly the same as the behaviour coded in the analogue core. It has been coded as a new 'RSD' GBS – with a new code template – and a new hierarchical double comparator GBS, as shown in the functional diagram of figure 4. The interface is modelled using usual hierarchical input and output GBSs and a current balance-sheet. These interface constructs are described in more detail in (Moser et al., 1994).

4.3 Results

The different models have been simulated for a DC input voltage of 0.7V and the results are compared with the simulation of an 8-bit CMOS circuit implementation developed in (Heubi et al., 1996). The conditions of the experiment are summarized in table 1.

Table 1 Settings for the RSD converter parameters

Sample freq.	*Numb. of bits*	*Clock freq.*	*Ref. voltage*	*Supply voltage*
16 kHz	8	128 kHz	1.3 V	±1.3 V

We show in figure 5 the shape of the curves obtained for one conversion with the various models.

Obviously, the primary behaviour – the result of the comparison – is the same. To interpret the value of the output sample, the n-bit binary number that corresponds to V_{outn} is subtracted from the one that corresponds to V_{outp}. This difference must then be divided by the signal dynamics ($2n$) and multiplied by the reference voltage V_r, giving

$$\frac{128+16-8+2}{256} \cong 0.054 \cong \frac{0.7}{1.3}.$$

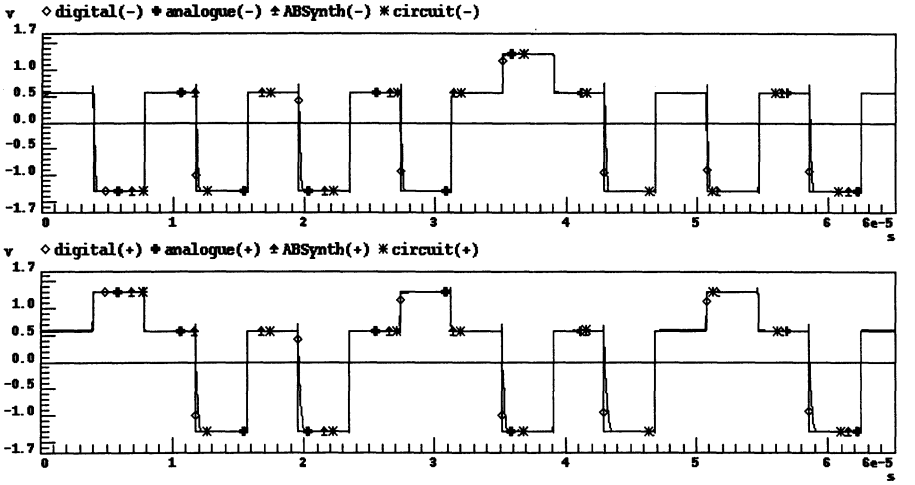

Figure 5 Output signals of the RSD converter models.

Table 2 presents some interesting figures about code size and simulation time.

Table 2 Code size and simulation time (one conversion) with ELDO on Sun Sparc 10/30

Model	*Lines*	*States*	*Signals*	*Variables*	*Simul. time*
analogue	200	11	0	13	13s
digital	212	8	5	14	15s
ABSynth	906	126	0	28	51s
CMOS					6 min

The analogue and digital models are quite similar in size and in simulation speed. The ABSynth model, however, is much bigger, which leaves room for optimization, as explained in (Moser, 1996). The ABSynth model is also slower and the simulation time seems to be linked to code size, especially to the number of states. Anyway, all

the HDL-A models are faster than the Spice-like description of the CMOS implementation.

5 CONCLUSIONS

In this paper we addressed the problem of modelling sampled-data systems with a mixed-mode hardware description language. Analogue and digital modelling solutions were proposed for typical basic operations like sampling and shifting and for the coding of difference equations as well as elementary z-domain cells. The analogue modelling solutions could also be implemented with ABSynth. All model variants were validated on a theoretical filter example and on a 'real-life' design case.

The language we used, HDL-A, showed enough flexibility for our purpose. Coding in the digital part was quite obvious while coding in the analogue part was somewhat more trickier but 'analogue' models were finally found to be as fast as 'digital' ones. However, we can assume that the digital simulation engine of HDL-A is not optimized and that better results could be achieved by 'digital' sampled-data modelling with a better digital simulator. Furthermore, it would also be possible to model nearly all-digital models using digital terminals (PORTs). Such models would be faster than analogue ones but they would include less information about the interface's behaviour.

In the future, it would be nice to write portable models using the standard language VHDL-AMS instead of a proprietary one (HDL-A). Probably, VHDL-AMS will offer as much analogue modelling functionality as HDL-A. An exception will certainly be the function previous(x) because it is closely related to the implemented solving algorithm. Furthermore, as the whole VHDL syntax will be available in the digital part, it will be possible to code more complex mixed analogue-digital models.

We can also assume that ELDO is not the fastest solution to simulate mainly discrete systems such as sampled-data systems. Indeed, ELDO calculates analogue time points not only when clock events arise but also between them, depending on the analogue time-step management. This will slow down the simulation but can also be seen as an advantage when some analogue timing behaviour – e.g. rising slope – is of interest.

The ABSynth models were slower than 'hand-coded' ones but still faster than transistor-level descriptions. Consequently, we consider that ABSynth, although not designed for sampled-data modelling, can still be used in this context. An ideal solution, yet to be developed, would be the coupling of ABSynth with a digital modelling tool.

6 ACKNOWLEDGEMENTS

This work was supported partly by the Swiss Foundation for Microtechnology Research (FSRM) and partly by the Swiss Priority Program MINAST (project MicroSIM). We would also like to thank Louisa Grisoni and Alexandre Heubi for their help in modelling the RSD converter.

7 REFERENCES

ANACAD Electrical Engineering Software (1994) HDL-A User's Manual. Issue 1.

Ginetti, B., Jespers, P.G.A., and Vandemeulebroeke, A. (1992) A CMOS 13-b Cyclic RSD A/D Converter. *IEEE Journal of Solid-State Circuits*, Vol. 27, No. 7, 957-64.

Heubi, A., Balsiger, P., and Pellandini, F. (1996) Micro Power 'Relative Precision' 13 bits Cyclic RSD A/D Converter. *Proceedings of the IEEE International Symposium on Low Power Electronics and Design.*

Kuc, R. (1988) Introduction to Digital Signal Processing. McGraw-Hill Book Company, New York.

Kunt, M. (1984) Traitement Numérique des Signaux. Traité d'Electricité, Vol. XX, Presses Polytechniques Romandes, Lausanne.

Moser, V., Nussbaum, P., Amann, H.P., Astier, L., and Pellandini, F. (1994) A Graphical Approach to Analogue Behavioural Modelling. *Proceedings of the European Design and Test Conference,* 535-9.

Moser, V., Amann, H.P., Nussbaum, P., and Pellandini, F. (1995) Generating VHDL-A-like Models Using ABSynth. *Proceedings of EURO-DAC'95 European Design Automation Conference with EURO-VHDL'95,* 522-7.

Moser, V. (1996) Computer-Aided Behavioural Modelling of Analogue Systems. Thèse de doctorat, Université de Neuchâtel.

8 BIOGRAPHY

Vincent Moser received his degree of electrical engineer from the Swiss Federal Institute of Technology (EPFL), Lausanne, in 1990. From 1990 to 1992, he worked with ASCOM Business Systems AG, Solothurn as a development engineer in the area of digital telephony. In 1992 he joined the Institute of Microtechnology of the University of Neuchâtel as a research assistant. He received his PhD from this university in 1996 with a thesis entitled 'Computer-Aided Behavioural Modelling of Analogue Systems'. His current research interests include analogue design methods.

modelling and simulation of electronics, modelling and simulation of microsystems as well as hardware-software cosimulation.

Hans Peter Amann received his degree of electrical engineer from the Swiss Federal Institute of Technology, Zurich, and his PhD from the University of Neuchâtel in 1982 and 1990 respectively. Since 1990, he has been with the Electronics and Signal Processing group of the Institute of Microtechnology, University of Neuchâtel, where he is responsible for the activities in modelling and simulation. His research interests are oriented towards design automation and rapid prototyping of electronic circuits, covering the design path from graphical specification over model generation and simulation down to the implementation level. More recently, this domain of activity has been extended towards microsystem design.

Fausto Pellandini received his degree of electrical engineer and his PhD from the Swiss Federal Institute of Technology, Zurich, in 1962 and 1967 respectively. In 1972, he became Professor for electronics at the University of Neuchâtel, and was in charge of the foundation of the Institute of Microtechnology (IMT) which he headed until 1985. Since 1978, he has also been a lecturer with the Swiss Federal Institute of Technology, Lausanne, where he became part-time ordinary professor in 1989. The research activities of the group he leads at IMT include digital filtering, VLSI for signal processing, low-power VLSI, microsystem design, and high-level modelling and simulation.