Chapter 15

Code Generation From MATLAB – Simulink Models

Tiago da Silva Almeida, Ian Andrew Grout and
Alexandre César Rodrigues da Silva

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/57575

1. Introduction

Nowadays, computational tools are indispensable in the design of electronic circuits due to the increase in complexity of electronic circuits and the need to manage large amounts of related data to design. The development of new methodologies and tools has become a strategic area in the development of new technologies, in particular the development of CAD (Computer Aided Design) tools. CAD tools can be best understood as design information management systems, along with the creation of graphic based input and simulations of the designs are created. These simulations can be used, shared, published, republished and reused in different formats, scales and levels of detail.

But with so many studies involving different methodologies and computational tools, in [1], it was proposed to form a classification model for design and tools at the Electronic System Level (ESL) of design abstraction. This chapter details the characteristics and approaches that differ between computational tools of design, for modeling at level hardware and software. Figure 1 illustrates the design flow in different levels of abstraction. The design flow, called the Double Roof Model, is considered by [1] as an extended version of the Y diagram presented by [2][3]. However, there are still problems of incompatibility between designs the tools and this identifies a number of weaknesses.

Figure 1 shows the process of designing the top-down methodology in an ideal way. The left side corresponds to the process of software creation, while the right side corresponds to the process of hardware creation. Each side is divided into different levels of abstraction, e.g., Task and Instruction (software) or Components and Logic (hardware). There is a common level of abstraction, named ESL. In this level, we cannot distinguish between hardware and software. At each level, we can run a synthesis step (continuous vertical arrow) and the specification is transformed into an implementation. Horizontal dotted arrows indicate the steps which we

can change the models of individual elements in the implementation directly to the next level of abstraction (lower level abstraction).

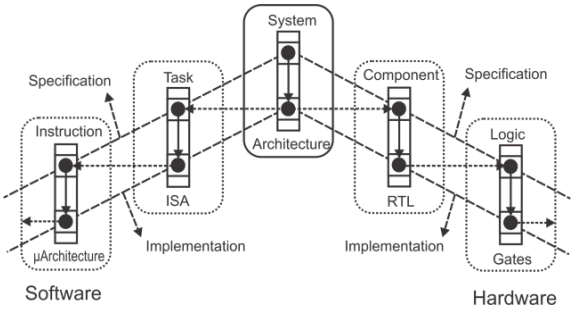

Figure 1. Flow design of electronic systems, also known as Double Roof Model.

In this chapter, we present a methodology for ESL designs based in four computational tools. The first one is the SF²HDL tool (Stateflow to Hardware Description Language or transition states table). The SF²HDL has the ability to convert models of finite state machine (FSM) in Stateflow / Simulink in a corresponding description in VHDL, Verilog or even a text file with a state transition table. The descriptions in the VHDL and Verilog languages are generated at the behavioral abstraction level for the finite state machine. The second computational tool, called MS²SV (MATLAB / Simulink to SystemVision), is able to generate descriptions in VHDL-AMS from models described in Simulink and all of model structure design for simulation in SystemVision environment from Mentor Graphics. The descriptions are generated in VHDL-AMS since even in the standard Simulink toolbox. there are many components with different signal representations of a purely digital representation. The third computational tool developed, called BD²XML (Block Diagram to XML), is able to generate a textual representation of the architecture of this model in XML markup language, also from models described in Simulink. XML was chosen for its applicability and it can be used as object code by other tools. The fourth and last computational tool described works together with BD²XML and it is similar to SF²HDL. The tool, called SF²XML (Stateflow to XML), is able to capture the relevant information in the Stateflow file and generate a corresponding description in XML. The file resulting from the conversion is according to the standard of XML structure called SCXML (StateChart XML) proposed by the W3C [4].

For acceptance in the community of software development, maintainability and practicality, we chose to develop the translation environment using an object-oriented approach. For robustness, the C++language was chosen. With C++ language is possible create linear and nonlinear data structure, and languages like Java, it is not possible do it. In our tools, we used liner lists to store dynamically the data in memory. Thus, C++language is a better choice for programming language. Another reason for this choice, it is the common features between methods were coupled to objects, so there was created a logical interaction with low representational gap.

http://dx.doi.org/10.5772/57575

As case study designs, the digital-to-analog converters (DAC) design was used to explain the MS²SV and BD²XML. Data converters are circuits that transform a given representation to another. The ADC (analog to digital converter) is used to convert analog signals to digital data. The digital to analogue converter works in the opposite manner to the ADC, converting digital data input to analogue signal output proportional to value of input digital.

Considering the DAC operation, *n* binary input bits (which can be considered as representing the binary code of a decimal value) are received and there are 2^n possible combinations of binary input. There is also an additional input to the circuit design that is used as a reference signal, represented by V_{ref} and the reference is a voltage level, which is used to determine the maximum value that converter can generate on its output. The analog value is generated by the weighted sum of *n* inputs, multiplied by the reference voltage. Inputs are weighted according to magnitude of each bit, where *n* is the magnitude of input bit, *x* is total number of inputs, and *b* is DAC input value contained in the bit of *n* magnitude, where $b_n \in \{0,1\}$, as described in:

$$V_P = \sum_{n=1}^{x} \frac{1}{2^n} b_n \tag{1}$$

Using Equation (1), the analogue output voltage from the data converter is obtained by multiplying the result of Equation (1) by the reference voltage to obtain:

$$V_{out} = V_P \times V_{ref} \tag{2}$$

There are several methods possible to implement the DAC operation (i.e., there are several different DAC design architectures possible). One common method used is R/2R ladder circuit, where only two values for resistors in circuit are used (R and 2R), and output current depends on positions of switches that are controlled by inputs [5].

To explain the use of SF²HDL and SF²XML, we used a HDB3 (High Density Bipolar 3) finite state machine [6]. HDB3 line code, which is a technique for detecting synchronism with bipolar signaling, and it has been used over the years in digital systems. A finite state machine can be represented by state diagrams. State diagrams are used to graphically represent a state transition functions set. Having an input event set, the output event set and the next state set can be determined.

There are also several other forms to represent finite state machine that differ slightly and have different semantic. The State Transition Table and Petri Net are two examples. Figure 2 shows generic schematic of a finite state machine. The circuit has a finite number of inputs, representing all input variable set $\{N_1, N_2, \ldots, N_n\}$. Thus, the circuit has a finite number *m* of outputs, determined by the output set of variables $\{M_1, M_2, \ldots, M_m\}$.

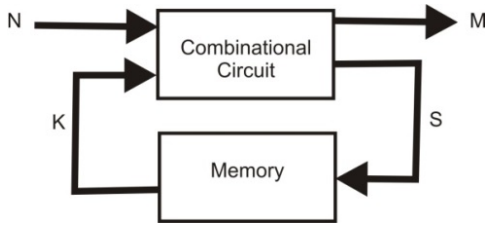

Figure 2. Schematic of a Finite State Machine.

The value in each memory element is called a state variable, and the set denoted by $\{K_1, K_2, \ldots, K_k\}$ form the state variable set. The values contained in k memory elements define the internal state of finite state machine. The outputs $\{M_1, M_2 \ldots, M_m\}$ and internal transition functions $\{S_1, S_2, \ldots, S_s\}$ depend on external inputs (input functions) and internal states (state functions) of state machine and they are defined by combinational circuits. The values S, which appear in the role of state transitions of machine at time t, determine values of state variables at time $t+1$ and therefore define the next state of machine.

The state transition diagram or the state transition table makes the relationship between present state, next state, input and output.

The finite state machines that determine the next state $K(t+1)$ only in the present state $K(t)$ and present input $N(t)$ are called deterministic machines or Markovian. Deterministic machines can be represented by the following equation:

$$K(t+1) = f[K(t), N(t)] \tag{3}$$

Where, f is a state transition function. The value of output M(t) is obtained through of two ways:

$$M(t) = g[K(t)] \tag{4}$$

or,

$$M(t) = g[K(t), N(t)] \tag{5}$$

where, g is an output function.

A finite state machine with properties described in Equations (3) and (4) is called Moore model and a machine described by equations (3) and (5) is called Mealy model [7].

To illustrate the state transition diagram, Figure 3 shows a state transition diagram of HDB3 code line, in hexadecimal representation as proposed in [6]. The encoder needed three parts

http://dx.doi.org/10.5772/57575

of memory to store and to identify four successive 0. After this, it was necessary to use a flip-flop to store the polarity of the pulse (0 for+or 1 for-), and finally other flip-flop to store the polarity of the violation. The encoder requests a minimum of 25=32 states. The state 00 represents the initial state of the machine, and the two pairs of state 07 and 0F (in the left) and 1F and 17 (in the right) represent the description AMI, what causes the alternate code between +1 and-1 [6].

2. Related research

Project-level electronic systems, also known as ESL (Electronic System Level), require much effort designer. The first work to classify the steps involved in system design at a high level of abstraction [2] subsequently refined to a more current [1] and [3].

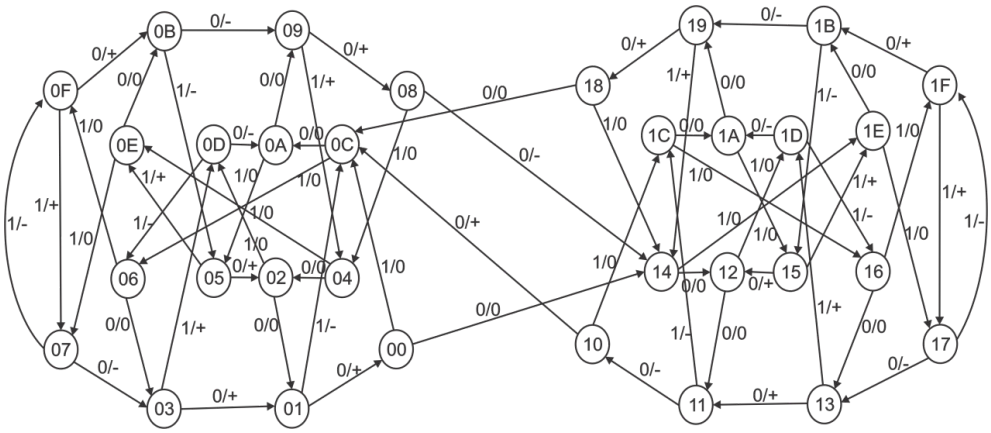

Figure 3. State transition diagram for HDB3 line code.

Some studies deal with design optimization or minimization, for example the work in paper [8]. This work explored the computational complexity of logic minimization problem on two levels for digital circuit designs. In [9] was developed a genetic algorithm for synthesis of multi-core architectures. The target system was an implementation of Petri net. In [10] was presented a method of re-engineering the FSM by rebuilding it with a different topology, but with equivalent functionality. The method is mainly concerned with power consumption of the finite state machine and uses a genetic algorithm to find the best solution. This approach facilitates the process of synthesis for optimal implementations of the system.

The representation of electronic systems by means of the FSM facilitates verification of the system and formal methodologies can be developed to assist in the synthesis process. In [11] was presented a tool capable to translate a FSM described in the VHDL. The focus of the work was modeling and the verification. In [12] was developed a conceptual mathematical model

for automatic checking of the FSM by observing the outputs of this particular FSM, allowing checking faulty behavior.

Hardware description languages (HDLs) are widely used in the design of electronic systems and also represent a major means for simulation and project documentation. In [13] was developed an algorithm for automatic generation of analog circuit models for modeling faults in the circuit at a high level. In [14] was developed a methodology for implementing FFT (Fast Fourier Transform) processors in reprogrammable devices using VHDL. In [15] was developed the architecture for implementing a SPICE (Simulation Program with Integrated Circuit Emphasis) simulator using an FPGA (Field Programmable Gate Array)

In [16] the authors proposed a simulator for the VHDL-AMS (Analog and Mixed Signal) language developed in the MATLAB environment. The tool was compared with others simulators available commercially. Another example that can be cited is the tool called SF2VHD, presented in [17]. That tool is capable to translate hierarchical models of finite states machine, modeled in the Stateflow environment, in a code corresponding structural VHDL.

In the work described in [18] was developed a tool called Sim2HDL, which accomplishes the automatic translation of Simulink models in a hardware description language in the attempt to drastically reduce the project time. The generated language can be VHDL or Verilog, at a behavioral level description, and the system can be implemented in FPGAs using commercial synthesizers.

Some commercial modeling tools incorporate the translation need for different forms of modeling. An example is a MATLAB C Compiler tool, developed by Mathworks. That tool translates programs written in MATLAB for a corresponding program in the C language. The functionality and main advantages of these tools are described in [19].

In [20] was proposed an approach to automatic synthesis of additional decoders. The method attempts to find and remove cases where there are no equivalent decoders. To discover all decoders that can exist simultaneously, an algorithm based on functional dependency has also been proposed. To select the correct decoder another algorithm is used to infer formula of each decoder precondition.

In [21] was developed a framework for the synthesis of electronic systems at a high level based on abstract finite state machines. The framework called synASM is capable of generating a CDFG (Control Data Flow Graph) from hardware descriptions based on C language. The authors extended the definitions of abstract state machines to support the parallelism and timing. After generating the CDFG extended, it was possible the automatic generation of optimizable hardware descriptions in VHDL and implementation in a FPGA.

In [22] was presented a complete approach decoder for communication systems. The modelling was based on finite state machines and the method was capable of identifying whether decode exists or not, by observing the input sequence and output of encoder. [23] presented a method of representation and synthesis of Boolean expressions recursively. The method performs a sequence of operations with denial implications using two memristors. With the method, it was proved that it was possible to reduce the number of necessary implications for

http://dx.doi.org/10.5772/57575

the expression representation and inference using multiple inputs can reduce the computation of memristor implementation.

In [24] was proposed an algorithm for identifying flowcharts structure. Twelve structures were identified and then the algorithm was used to generate the code flowchart identified using recursion. The technologies and algorithms were used in an integrated development platform. [25] presented an approach of output bit selection based on counter for output response compaction in test systems observable. The hardware implementation requires only a counter and a multiplexer. Thus, the complexity was reduced and the control area was simplified. Furthermore, a change in the ATPG (Automatic Test Pattern Generation) tool was not required. Two output selection algorithms have been developed in several operations which counters can be employed.

In [26] was discussed some work and the importance of developing CAD tools for the synthesis and design of systems and multi-core architectures, highlighting the importance of multi-core optimized and efficient resource allocation. [27] presented revised performance measures of some data type converters ADC (Analog-to-Digital Converter), such as SNR (Signal to Noise Ratio), SINAD (Signal to Noise Ratio and Distortion), SFDR (Spurious free Dynamic Range), THD (Total Harmonic Distortion) and ENOB (Effective Number of Bits). These metrics were analyzed with the case study of an ideal 14-bit converter in MATLAB and a converter with 12-bit commercial manufactured by Analog Devices.

In [28] was presented a methodology for genetic optimization based on VHDL-AMS for Fuzzy Logic controllers. The fuzzy logic controller was used for modeling work in automotive systems in mixed physical domain. A genetic algorithm was developed and simulated in SystemVision environment, is employed in the generation of rough fuzzy sets.

3. Generation code methodology of SF²HDL

The SF²HDL is a computational tool capable to convert a state transition diagram into a hardware description language description. This diagram is described in the Stateflow environment [29]. The Stateflow is toolbox inside Simulink is used to simulate finite state machines in different contexts. The SF²HDL tool is a simplified graphical interface that allows the user to select between generate (i) an input file to be processed by TABELA program and (ii) a.*vhd* file with a behavioral VHDL description obtained directly from state transition diagram.

The TABELA program was developed for synthesize of a finite state machine and was implemented in Pascal language by researchers from UNICAMP [7] (Universidade Estadual de Campinas-Brazil). The program generates combinational functions that implement the finite machine described by state transition table. The transition functions and output functions are minimized and the cost of minimization using minterms and maxterms [7]. The transitions are specified on table form using one line per state transition, so: present state, next state, input, output. The description end is represented by notation "-100". Figure 4 shows the standard file for TABELA program input.

The states, the inputs and outputs must be in decimal notation. The finite state machines can be completely or incompletely specified.

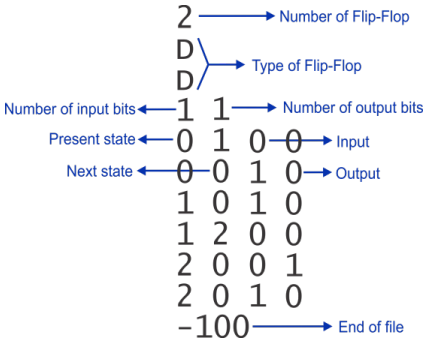

Figure 4. Input file for TABELA program.

The TABELA program assembles a state transition table and stores it in output file. From this table are obtained minterms, maxterms and don't care states of transition functions of all internal flip-flops and circuit output functions. Using Quine-McCluskey algorithm, is made the minimization of Boolean functions.

Figure 5 illustrates the main interface of SF²HDL program. The SF²HDL has a menu called **Ferramentas** (Tools) with the following options: a) **Nova Tradução** (Translation), used to open the file with a finite state machine modeled in Stateflow environment, b) **Traduzir** (Translate), which performs the model translation into one of language was chosen, c) **Sair** (Quit), which terminates the program, and d) **Sobre** (About) option that displays a screen with a brief comment on the tool developed.

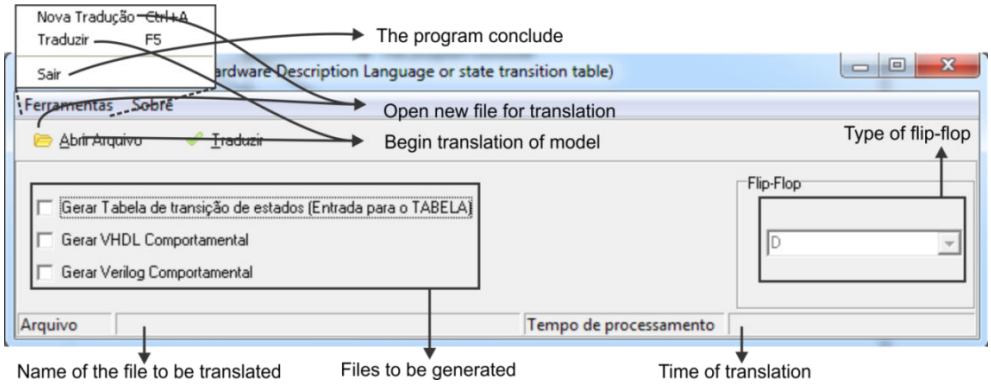

Figure 5. Main interface of SF²HDL program.

http://dx.doi.org/10.5772/57575

Thus, from all the information found in the.*mdl* (Simulink file) file, it is possible to determine the present state, the next state, the input and output of the finite state machine. According to the structure of each file, all the information is grouped and then is generated the input file for TABELA program and/or the corresponding behavioral VHDL. Figure 6 presents the functional diagram of SF²HDL program with all necessary steps to perform the translation, as previously reviewed.

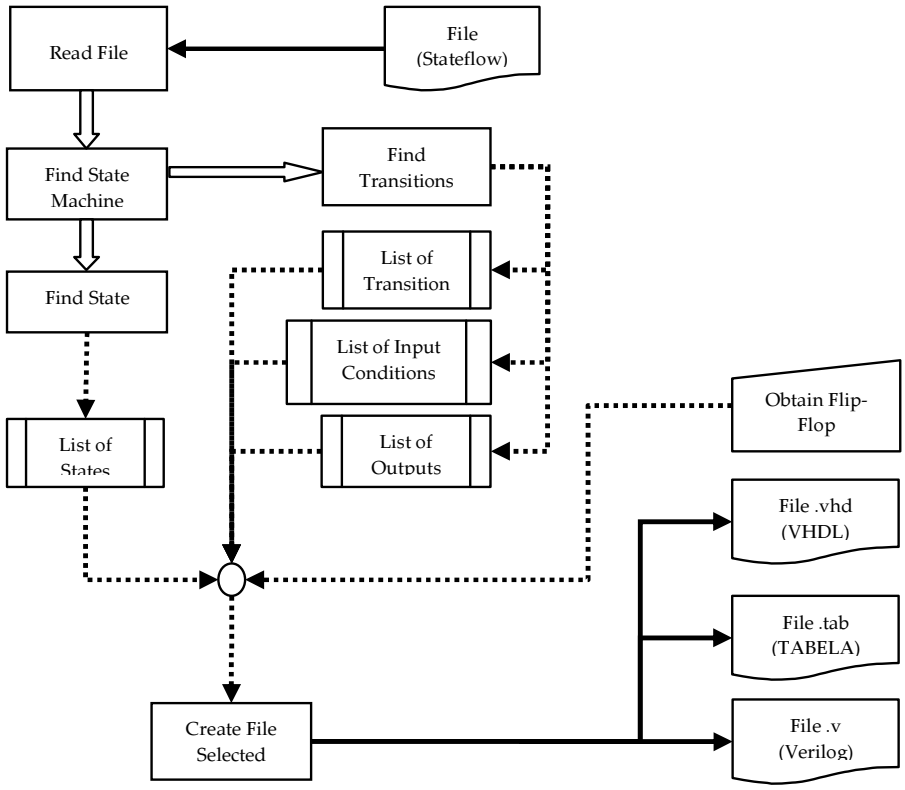

Figure 6. Block diagram of SF²HDL.

SF²HDL initially reads the file containing finite state machine, simulated in the Stateflow environment. Then, the program finds the state machine structure and it makes a temporary storage of this information. After that, the state location list for all states is generated and through the transition locations is generated a transitions list, identifying the conditions for the inputs and outputs. When the user selects to generate the input file for the TABELA program, there is the need to choose what type of memory element (flip-flop) will be used in the generated file. The memory element type must be put manually by the user.

Figure 7 shows the class diagram of SF²HDL. The SF²HDL was built with many inheritances between the classes. There are two objects to store the structure found in the.*mdl* file; the state information and transition information. This distinction is considered because the differences between a Mealy machine and Moore machine described in Equations (3), (4) and (5). SF²HDL is capable of recognizing which machine is used in the model. The **fsm** object is responsible to read the.*mdl* file and to save this information. The **misc** has some operations used many times during running of SF²HDL. The **create_files** object is responsible to create the file that was chosen by user, as illustrate in Figure 7. As SF²HDL can create three files at the same time, the **main** class call the **create_files** more than once, if it is the case.

3.1. Simulation results of HDB3 line code and conversion results

For modeling the state transition diagram on Stateflow environment, some rules must be obeyed, such as the name of the state contained within the symbol should never begin with numeric character (0, 1, 2, 3, 4, 5, 6, 7, 8 or 9), and the condition in the transition to the Mealy model must meet the following syntax:

$$[e[t] == 0]\{s = 0;\} \tag{6}$$

where e represents the variable that will receive input value, t is a variable that represents a discrete time and s is an output variable. The external bracket represents the input condition in the transition from one state to another to obtain an output. The double equal signs (==) represent the verification of a condition and the equal sign alone (=) represents the allocation of value [30].

For proper functionality of the SF²HDL program, is necessary that the values in the transition of model in Stateflow environment, both at input and output, are decimal representations. Figure 8 presents the state diagram of HDB3 code on Stateflow environment.

In simulation are used several words with signs and bits from varying sizes. However, it was standardized in using an input word in the simulation of 24 bits being formed by the following bits: 000100110000110101000100. We used this bit stream because with it we can simulate both characteristics of signal encoder, AMI and HDB3, in a real situation. The result of the simulation is presented in the Figure 9.

The TABELA program creates a new file, containing a description of Boolean functions from the circuit already minimized.

Listing 1 was reduced and is shown only important parts of minimization. Each Boolean function in the finite state machine is minimized. As HDB3 has five flip-flops and two output bits, it is possible to see seven functions in Listing 1 (lines 1, 11, 21, 28, 34, 40 and 51). In each function are described the function minterms in decimal, e.g. line 3. The function prime implicants and essentials are shown too, e.g. lines 4-9. And the function cost is shown in lines 10, 20, 27, 33, 39, 50 and 58. The total cost of the circuit is shown in lie 59, for HDB3 case is 95. It is important to highlight that the cost is expressed like sum of product by TABELA.

http://dx.doi.org/10.5772/57575

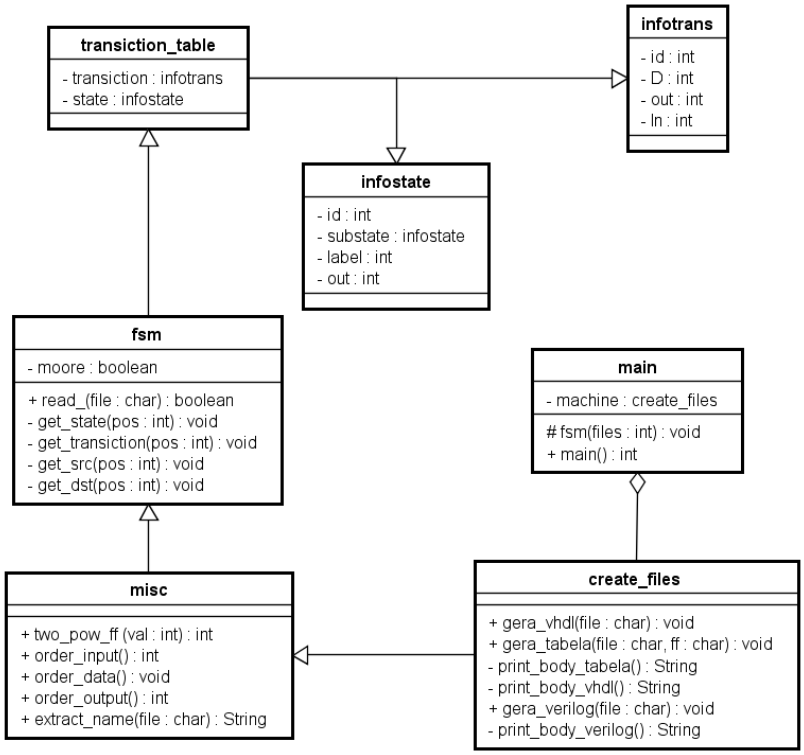

Figure 7. Class diagram of SF²HDL.

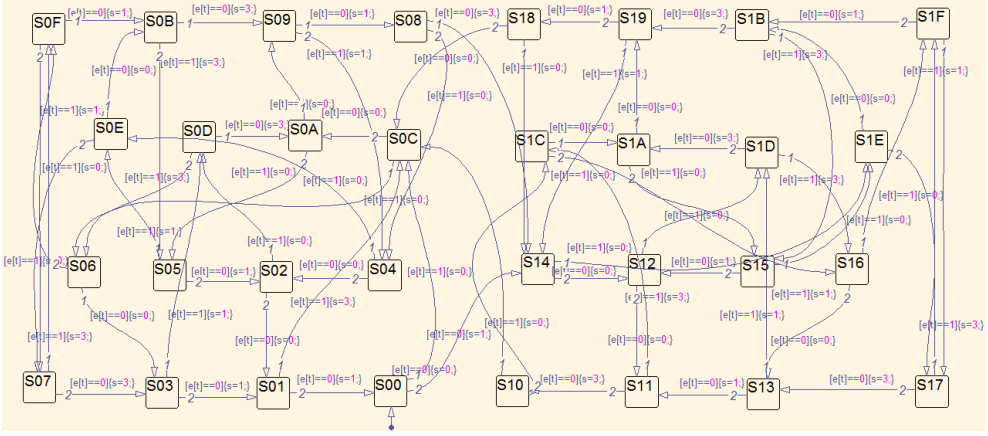

Figure 8. Description of HDB3 code on Stateflow.

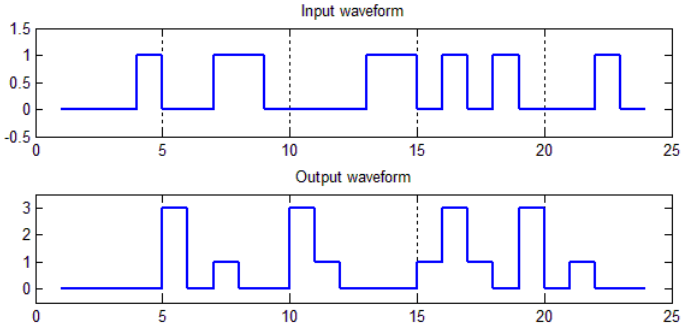

Figure 9. Results of simulation of HDB3on Stateflow.

```
1.  FUNCAO D4
2.  =========
3.  MINTERMOS : 0; 8; 48; 17; 49; 18; 50; 19; 51; 20; 52; 21; 53; 22; 54; 23; 55; 25; 57; 26;
    58; 27; 59; 28; 60; 29; 61; 30; 62; 31; 63;
4.  IMPLICANTES PRIMOS ESSENCIAIS :
5.  ESSENCIAL: 20  REDUNDANCIA:  43 -> X1X1XX
6.  ESSENCIAL: 18  REDUNDANCIA:  45 -> X1XX1X
7.  ESSENCIAL: 17  REDUNDANCIA:  46 -> X1XXX1
8.  ESSENCIAL: 48  REDUNDANCIA:  15 -> 11XXXX
9.  ESSENCIAL: 0   REDUNDANCIA:   8 -> 00X000
10. CUSTO FINAL DE D4 = 18
11. FUNCAO D3
12. =========
13. MINTERMOS : 32; 33; 34; 35; 36; 37; 38; 39; 9; 10; 11; 12; 13; 14; 15; 48; 16; 49; 50; 51;
    52; 53; 54; 55; 24; 25; 26; 27; 28; 29; 30; 31;
14. IMPLICANTES PRIMOS ESSENCIAIS :
15. ESSENCIAL: 16    REDUNDANCIA:    8 -> 01X000
16. ESSENCIAL: 12    REDUNDANCIA:   19 -> 0X11XX
17. ESSENCIAL: 10    REDUNDANCIA:   21 -> 0X1X1X
18. ESSENCIAL: 9     REDUNDANCIA:   22 -> 0X1XX1
19. ESSENCIAL: 32    REDUNDANCIA:   23 -> 1X0XXX
20. CUSTO FINAL DE D3 = 21
21. FUNCAO D2
22. =========
23. MINTERMOS : 32; 0; 33; 34; 35; 36; 37; 38; 39; 40; 8; 41; 42; 43; 44; 45; 46; 47; 48; 16;
    49; 50; 51; 52; 53; 54; 55; 56; 24; 57; 58; 59; 60; 61; 62; 63;
24. IMPLICANTES PRIMOS ESSENCIAIS :
25. ESSENCIAL: 32    REDUNDANCIA:   31 -> 1XXXXX
26. ESSENCIAL: 0     REDUNDANCIA:   56 -> XXX000
27. CUSTO FINAL DE D2 = 6
28. FUNCAO D1
29. =========
30. MINTERMOS : 4; 36; 5; 37; 6; 38; 39; 7; 44; 12; 13; 45; 46; 14; 47; 15; 20; 52; 21; 53; 22;
    54; 55; 23; 60; 28; 61; 29; 62; 30; 63; 31;
31. IMPLICANTES PRIMOS ESSENCIAIS :
32. ESSENCIAL: 4     REDUNDANCIA:   59 -> XXX1XX
33. CUSTO FINAL DE D1 = 1
34. FUNCAO D0
35. =========
36. MINTERMOS : 34; 2; 3; 35; 6; 38; 39; 7; 42; 10; 11; 43; 46; 14; 47; 15; 18; 50; 51; 19; 22;
    54; 55; 23; 26; 58; 59; 27; 62; 30; 63; 31;
37. IMPLICANTES PRIMOS ESSENCIAIS :
38. ESSENCIAL: 2     REDUNDANCIA:   61 -> XXXX1X
39. CUSTO FINAL DE D0 = 1
40. FUNCAO Z1
41. =========
42. MINTERMOS : 33; 39; 7; 8; 11; 43; 13; 45; 17; 49; 55; 23; 59; 27; 61; 29;
43. IMPLICANTES PRIMOS ESSENCIAIS :
44. ESSENCIAL: 17    REDUNDANCIA:   32 -> X10001
45. ESSENCIAL: 13    REDUNDANCIA:   48 -> XX1101
46. ESSENCIAL: 11    REDUNDANCIA:   48 -> XX1011
47. ESSENCIAL: 8     REDUNDANCIA:    0 -> 001000
48. ESSENCIAL: 7     REDUNDANCIA:   48 -> XX0111
49. ESSENCIAL: 33    REDUNDANCIA:   16 -> 1X0001
50. CUSTO FINAL DE Z1 = 34
51. FUNCAO Z0
52. =========
53. MINTERMOS : 1; 33; 3; 35; 5; 37; 39; 7; 8; 41; 9; 11; 43; 13; 45; 47; 15; 16; 17; 49; 51;
    19; 21; 53; 55; 23; 57; 25; 59; 27; 61; 29; 63; 31;
54. IMPLICANTES PRIMOS ESSENCIAIS :
55. ESSENCIAL: 16    REDUNDANCIA:    1 -> 01000X
56. ESSENCIAL: 8     REDUNDANCIA:    1 -> 00100X
57. ESSENCIAL: 1     REDUNDANCIA:   62 -> XXXXX1
58. CUSTO FINAL DE Z0 = 14
59. CUSTO TOTAL DAS 7 FUNCOES = 95
```

Listing 1. Minimization generate by TABELA program for HDB3 study case

http://dx.doi.org/10.5772/57575

After the minimization by TABELA was used the TAB2VHDL program [31] which generated VHDL code in RTL (Register Transfer Level) that is shown in Listing 2. The TAB2VHDL is a tool developed by researchers in UNESP (Universidade Estadual Paulista-Brazil) in C language. The only task TAB2VHDL is generated the VHDL code from TABELA output.

```
1.  ENTITY HDB3 IS
2.  PORT(
3.  CLK, CLR: IN BIT;
4.  X0: IN BIT;
5.  Z0, Z1: OUT BIT
6.  );
7.  END HDB3;
8.  ARCHITECTURE RTL OF HDB3 IS
9.  SIGNAL VE0, VE1, VE2, VE3, VE4: BIT;
10. SIGNAL D4, D3, D2, D1, D0: BIT;
11. BEGIN
12. PROCESS (CLK, CLR)
13. BEGIN
14. IF CLR = '0' THEN
15. VE0 <= '0';
16. ELSIF CLK'EVENT and CLK = '1' THEN
17. VE0 <= D0;
18. END IF;
19. END PROCESS;
20. ...
21. ...
22. D4 <= ((VE4) AND (VE2)) OR ((VE4) AND (VE1)) OR ((VE4) AND (VE0)) OR ((X0) AND (VE4)) OR
    (NOT(X0) AND NOT (VE4) AND NOT (VE2) AND NOT (VE1) AND NOT(VE0));
23. D3 <= ( NOT(X0) AND (VE4) AND NOT(VE2) AND NOT(VE1) AND NOT(VE0)) OR (NOT(X0) AND (VE3) AND
    (VE2)) OR (NOT(X0) AND (VE3) AND (VE1)) OR (NOT(X0) AND (VE3) AND (VE0)) OR ((X0) AND
    NOT(VE3));
24. D2 <= ( (X0)) OR (NOT(VE2) AND NOT(VE1) AND NOT(VE0));
25. D1 <= ( (VE2));
26. D0 <= ( (VE1));
27. Z1 <= ( (VE3) AND (VE2) AND NOT(VE1) AND (VE0)) OR ((VE3) AND NOT(VE2) AND (VE1) AND (VE0))
    OR (NOT(X0) AND NOT(VE4) AND (VE3) AND NOT(VE2) AND NOT(VE1) AND NOT(VE0)) OR (NOT(VE3) AND
    (VE2) AND (VE1) AND (VE0)) OR (NOT(VE3) AND NOT(VE2) AND NOT(VE1) AND (VE0));
28. Z0 <= ( (VE3) AND (VE2) AND NOT(VE1) AND (VE0)) OR ((VE3) AND NOT(VE2) AND (VE1) AND (VE0))
    OR (NOT(X0) AND NOT(VE4) AND (VE3) AND NOT(VE2) AND NOT(VE1) AND NOT(VE0)) OR (NOT(VE3) AND
    (VE2) AND (VE1) AND (VE0)) OR (NOT(VE3) AND NOT(VE2) AND NOT(VE1) AND (VE0));
29. END RTL;
```

Listing 2. VHDL code generate by TAB2VHDL program for HDB3 study case.

In Listing 2, there is clear distinction in architecture between the flip-flops and the Boolean function to describe the behavior of HDB3. In lines 15-22 are described the flip-flop used (noting that this listing was reduced to presentation in this chapter). Lines 25-31 show the Boolean function of HDB3. This kind of abstraction has better results for synthesis in order to implement the circuit, e.g. in a FPGA (Field Programming Gate Array).

The other option is to generate the behavioral VHDL directly without using TABELA program. In Listing 3 is shown the file with behavioral VHDL code generated.

This description is different between Listing 2 as it is expressed at a behavioral level of abstraction for HDB3. In lines 3 and 4 in the entity, the signal representation is in decimal instead binary. This is a different way that VHDL can support the representation of input and output signals. Lines 14-41 express the behavior of the finite state machine. The reset signal is shown in line 18 and 19 and lines 21-40 show the transitions of HDB3. SF²HDL does not use the same label used in Stateflow, such as Figure 8. This case study coincidentally is the same (decimal representation instead hexadecimal), but in other cases then this label will be changed. The last process in description, lines 43-47, describes the update of state by clock signal.

The third description language generated by SF²HDL is a behavioral Verilog code [32], which it is shown in Listing 4.

```
1.  ENTITY Mealy_HD IS
2.  PORT(
3.  ent: IN INTEGER RANGE 0 TO 1;
4.  sai: OUT INTEGER RANGE 0 TO 3;
5.  clk, clr: IN BIT
6.  );
7.  END Mealy_HD;
8.  ARCHITECTURE lpssd OF Mealy_HD IS
9.  TYPE tipo_est IS (S0, S1, S2... S31);
10. SIGNAL est_atual, prox_est: tipo_est;
11. BEGIN
12. behavior: PROCESS (reset, input, state)
13. BEGIN
14. nxtstate <= state;
15. output <= 0;
16. IF reset = '0' THEN
17.  nxtstate <= S14;
18. ELSE
19.  CASE state IS
20.  WHEN S0 =>
21.   IF (input = 1) THEN
22.   output <= 1;
23.   nxtstate <= S12;
24.   ELSE
25.   output <= 1;
26.   nxtstate <= S2;
27.   END IF;
28.  WHEN S1 =>
29.   IF (input = 1) THEN
30.   output <= 0;
31.   nxtstate <= S12;
32.   ELSE
33.   output <= 0;
34.   nxtstate <= S2;
35.   END IF;
36.   ...
37.   ...
38.  END CASE;
39. END PROCESS;
40. clock: PROCESS
41. BEGIN
42. WAIT UNTIL clk'EVENT AND clk = '1';
43. state <= nxtstate;
44. END PROCESS clock;
45. END lpssd;
```

Listing 3. Behavioral VHDL code generate by SF²HDL for HDB3 study case.

Listing 4 is very similar to Listing 3, with the difference in the syntax between both languages, VHDL and Verilog. In the same way, lines 18-48 express the behavioral of the finite state machine, the reset signal is shown in line 21 and 23, lines 25-46 is shown the transitions and lines 50-52 describe the update of state by clock signal.

In Figure 10 is shown the simulation of behavioral VHDL code generated by SF²HDL on Quartus II environment [33], using the same word of data in the simulation as that in the Stateflow environment to verify accuracy and violation of HDB3 code too in the Quartus II.

There is a short delay in output least significant in Figure 10. But the delay does not disrupt the behavioral of the HDB3 encoder. The behavioral VHDL, RTL VHDL and behavioral Verilog codes were synthesized and implemented on the Cyclone II EP2C20F484C7 FPGA.

Figure 11 shows the simulation waveform of code generated by TAB2VHDL, i.e., using a RTL VHDL code. This is different from Figure 10 and in Figure 11 is possible to see a stranger behavioral of the code. Possibly this is due to some mistakes in TABELA program as the RTL VHDL is same of the optimization realized by TABELA program. Or even, possibly the results of synthesis of Quartus II. At the moment we do not know the cause of the problem.

Figure 12 illustrates the simulation results for behavioral Verilog code. The simulation waveform of Figure 12 is the same waveform in Figure 10, proving that this representation in behavioral Verilog and VHDL, because the level of abstraction in the same too. The synthesis result in FPGA Cyclone II is another equal result.

http://dx.doi.org/10.5772/57575

```
1.  module Mealy_HD (clk, in, reset, out);
2.  input clk, reset;
3.  input [0:0] in;
4.  output out;
5.  reg [1:0] out;
6.  reg [4:0] state;
7.  reg [4:0] nxtstate;
8.  parameter [4:0]
9.  S0 = 0,
10. S1 = 1,
11. ...
12. ...
13. S31 = 31;
14. always @ (in or reset or state) begin
15. nxtstate = state;
16. out = 0;
17. if (reset) begin
18.  nxtstate = S14;
19. end
20. else begin
21.  case (state)
22.  S0:
23.  if (in == 1) begin
24.   nxtstate = S12;
25.   out = 1;
26.  end
27.  else begin
28.   nxtstate = S2;
29.   out = 1;
30.  end
31.  S1:
32.  if (in == 1) begin
33.   nxtstate = S12;
34.   out = 0;
35.  end
36.  else begin
37.   nxtstate = S2;
38.   out = 0;
39.  end
40.  ...
41.  ...
42.  endcase
43. end
44. end
45. always @ (posedge clk) begin
46. state = nxtstate;
47. end
48. endmodule
```

Listing 4. Behavioral Verilog code generate by SF²HDL for HDB3 study case.

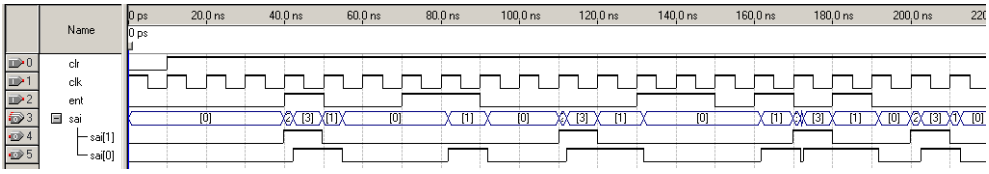

Figure 10. HDB3 line code Simulation in behavioral VHDL.

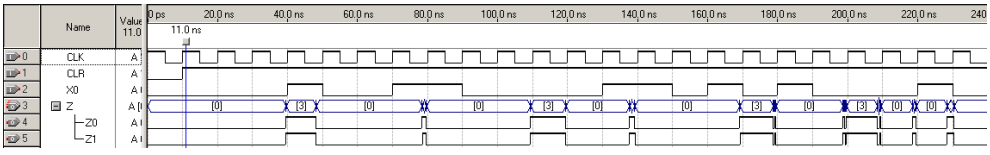

Figure 11. HDB3 line code Simulation in RTL VHDL.

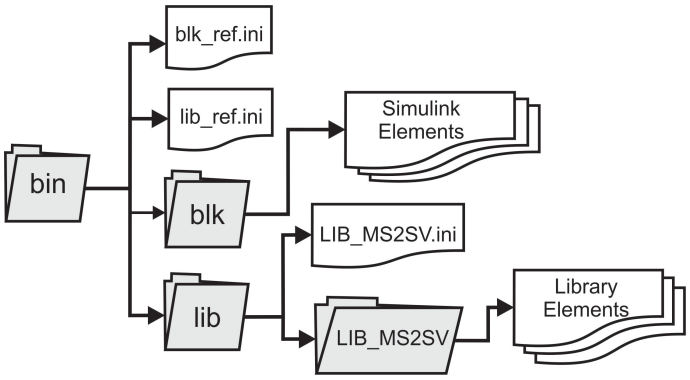

Figure 13. Directory structure and files that are used in configuration and the translation of models in MATLAB / Simulink.

4. Generation code methodology of MS²SV

The MS²SV is capable to convert a simulation model in Simulink to a description in VHDL-AMS [34]. This time, we used a VHDL-AMS description since the models can use different kinds of signals such as electrical, mechanical, hydraulic, etc. MS²SV has two key features:

1. The designer is able to add new components from the Simulink libraries. With this feature, the designer can use different elements previously defined in the standard Simulink libraries.

2. The addition of new Simulink libraries developed by the designer with more complex models, or to change the library named LIB_MS2SV also used in previously paper [36].

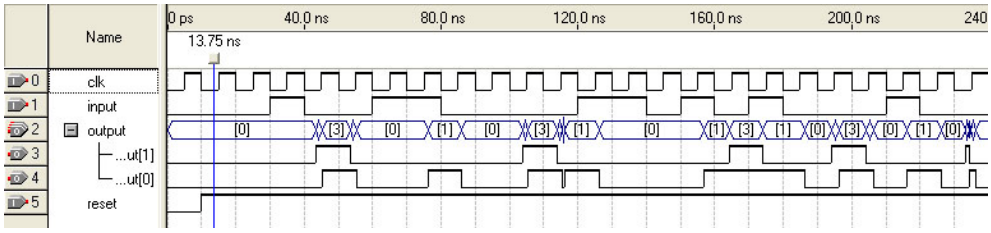

Figure 12. HDB3 line code Simulation in behavioral Verilog.

This added flexibility is made possible by creating a directory structure and configuration as shown Figure 13. From the root directory **bin** configuration files are split between configurations of elements from libraries of MATLAB / Simulink (**lib** directory) and elements of toolbox Simulink (**blk** directory) subject to translation. The pseudo-code concerning the VHDL-AMS models are also arranged between directories.

http://dx.doi.org/10.5772/57575

In the main interface, there is the menu containing the following options: a) **Arquivo** (File): In this option, there are two possible actions. With the first action, the user selects the Simulink model to be translated. When this action is selected, the status bar appears on the file name to be translated. The second action is a button to end execution of the tool, b) **Ferramentas** (Tools): In this option, the user is able to select the project location. The process of translation is then initiated. If there are unknown models that the tool is not able to recognize, it will display a message reporting the fact, c) **Configuração** (Configuration): In this option, there is an action called "Adicionar / Remover Componentes (Add / Remove Components)" and another called "Adicionar / Remover Bibliotecas (Add / Remove Libraries)". These options are used to add / remove elements of toolboxes and Simulink libraries developed by the designer, respectively. If there is not a model configuration to be translated, the tool is not able to recognize it and translate it.

The translation of the model starts with reading a Simulink generated file with an.**mdl** extension. Initially, a check is made of elements and libraries used in model. If there are no unknown libraries or elements, the translation process starts. The tool stores a list of components in model, the list of connections between these components and other important information for generation of circuit netlist. Then, project structure required for simulation and analysis in SystemVision is generated too [35][36]. Finally, all descriptions in VHDL-AMS and files for debugging for project needs are generated. Figure 14 shows the functional diagram of the MS²SV tool.

If subsystems are used, a VHDL-AMS model for each subsystem used is generated. The MS²SV enables the relationship between elements in the same library by creating hierarchical models.

Figure 15 represents the class diagram of tool MS²SV in a software perspective. All interface layers are represented since some aspects do not change the MS²SV functionality. In Figure 15, only the classes **Main**, **Save** and **VHDL** are in the interface layer. The others class makes the translation inference engine. The **SystemFile** class is responsible to generate all project structure to simulate in SytemVision environment. The **TranslateCode** is responsible to generate all VHDL-AMS files. This class inherits the features of **Checking** class, and **Checking** class checks if there is any primitive of Simulink, user libraries or subsystems. **Checking** inherits the features of **Lib** class and **Lib** class is responsible to catch the models into libraries. Last one, **Lib** inherits the features of **ReadMDL**, it responsible to reads the models in MATLAB / Simulink.

The translation process using elements of Simulink toolboxes is started with a verification of elements inside the model using **Checking** class operation. The **Checking** class uses the operation of **ReadMDL** class to read the Simulink file. The reading the Simulink file is done more than once because the interface that controls the inference engine allowing different file being loaded without translation. The process translate of elements inside libraries created by designer is quite similar to elements of Simulink toolboxes, but it includes the Lib class. Lib class copy the descriptions saved into internal libraries in tool MS²SV to represent libraries described by designer.

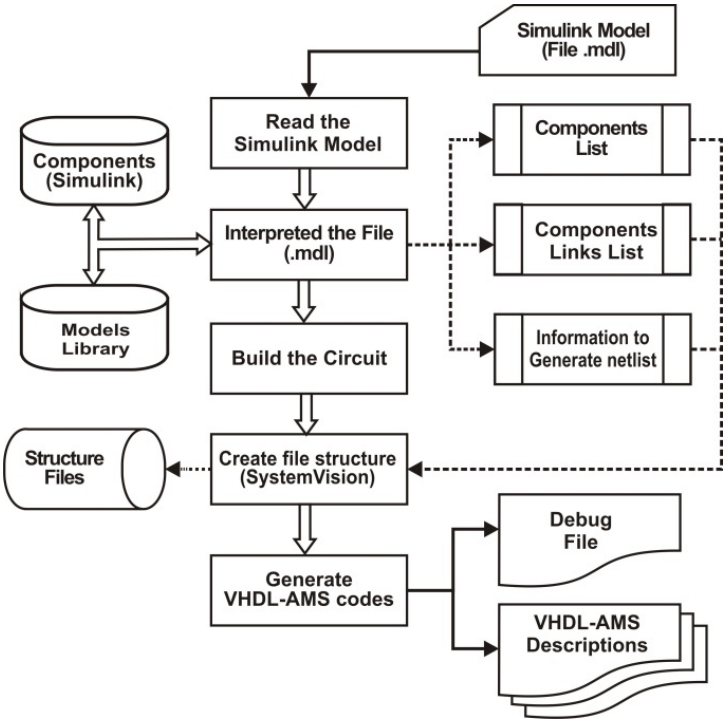

Figure 14. Functional diagram of the MS²SV tool illustrating the steps involved in translating the model to generate input file structure in SystemVision environment.

5. Case study of MS²SV and BD²XML

The DAC08, AD7524 and AD7528 DAC were chosen to model and simulate to evaluate the proposed methodology. All of them are monolithic data converter with 8 bit resolution used in applications such gain control circuit and stereo audio.

5.1. DAC08

DAC08 is a simpler operating model consisting basically of a converter with a resolution of 8 bit parallel input, which performs the digital to analogue conversion. The conversion of digital data into analog is done using the ladder R/2R where the binary inputs control the switching between the current arrival of resistors and current coming directly from the reference voltage.

http://dx.doi.org/10.5772/57575

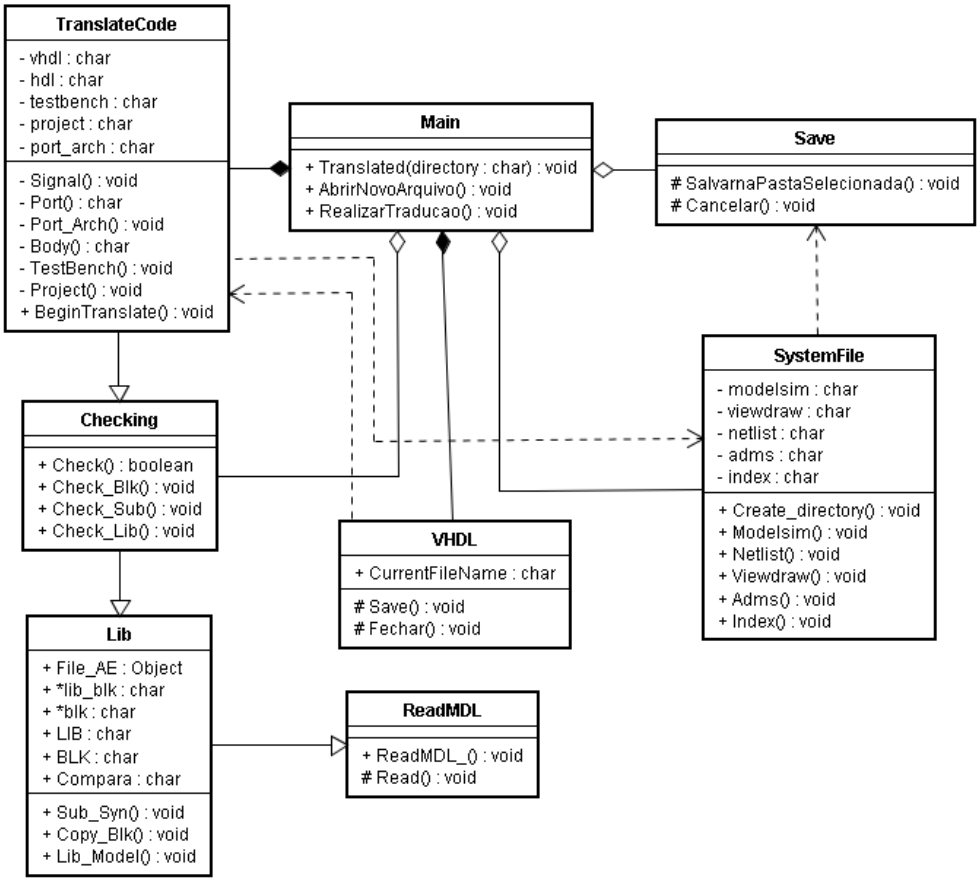

Figure 15. Class diagram of translations rules of MS²SV.

Based on the specifications found in the datasheet of DAC08 converter [37], it was possible to model it in MATLAB / Simulink. Other details of the implementation on a physical level, such as compatibility with TTL (Transistor-Transistor Logic) and CMOS (Complementary Metal Oxide Semiconductor) technology were not considered since the focus of this work was on modelling at high level of abstraction.

The creation of the R/2R Ladder subsystems involved the use of Equations (1) and (2) in order to adjust the weights of each bit, as shown in Figure 16 [38]. In Figure 16, the Ladder R/2R subsystem was constructed using basic components available in the Simulink libraries. The multiplication between the digital input and weights of each input was made with the "Gain components", the sum with seven "Sum components" of two values and multiplying the result by the reference voltage using "Product component". In this system there are two outputs, the analog signal and the difference between the output and the reference voltage.

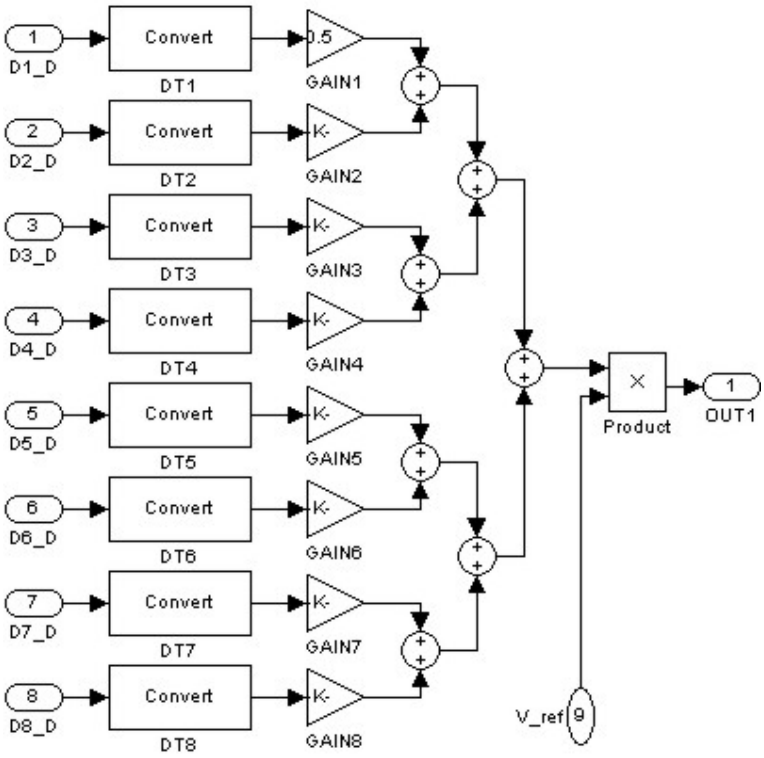

Figure 16. Subsystem in Simulink that represents the R/2R ladder using basic components of the Simulink toolbox, such as Product, Sum, Gain and Data Type Conversion.

An aspect that is important to emphasize in the creation of the R/2R ladder model shown in Figure 16 is that there are "Data Type Conversion components". These components are used to change the way representation of the binary signal (logic 0 or 1) for decimal representation (0 V or 1 V) and subsequent multiplication by weight. In MATLAB / Simulink, this component is not necessary. However, in VHDL-AMS, this component becomes important due to different ways of representing a signal in this language (i.e. the signal types used in VHDL and VHDL-AMS).

5.2. AD7524

The AD7524 has internally flip-flops of a latch type capable of storing last digital input, and a logic interface capable of controlling reading and storing digital input. The mode selection is controlled by CS_B and WR_B inputs. When CS_B and WR_B are at logic low (0) is enabled writing

http://dx.doi.org/10.5772/57575

mode, that is, the analog output representing the binary value at input bus DB_0-DB_7. But when CS_B or WR_B assumes a logic high (1), AD7524 is in hold mode, analog output has the value corresponding to last input in DB_0-DB_7 before WR_B or CS_B assume logic high. Table 1 shows the relationship of control inputs and selection mode of AD7524 [39].

CS_B	WR_B	Selection mode	Comments
0	0	Write	The output corresponds to activity on the input bus (DB_0 to DB_7).
1	X	Hold	The output corresponds to the last valid entry, stored in flip-flops.
X	1		

Table 1. Selection mode pins to control the AD7524.

Figure 17 represents the buffers used by the AD7524 to store last valid digital input. This component is basically formed by a set of latches and an input common to all latches to enable the output.

Figure 18 illustrates the circuit capable to represent a selection mode in Table 1. This circuit is minimum using only three logic gates (two NOTs and one AND) and Figure 19 shows a complete AD7524 with a circuit of selection mode, a latch buffer to store last input and Ladder R/2R.

5.3. AD7528

AD7528 is equivalent to two AD7524 converters in a single IC (integrated circuit). The data bus of the AD7528 is also numbered from DB_0 to DB_7. Each internal converter (DAC A and DAC B) has an individual reference pin. Both can vary ± 25 V. The AD7528 has only three control pins called DACA/DACB, CS_B and WR_B. The relationship between control pins is illustrated in the Table 2. At a moment when the AD7528 is in hold mode the last valid input is stored in buffers individually in each internal converter [35].

DACA/DACB	WR_B	CS_B	DAC A	DAC B
0	0	0	Write	Hold
1	0	0	Hold	Write
X	1	X	Hold	Hold
X	X	1	Hold	Hold

Table 2. Selection mode pin to control the AD7528 indicating write mode of the DAC A with all pins active low logic level or in write mode DAC B only with pin in DACA/DACB active-high level, any change in the CS_B pin or WR_B both converters are in hold mode.

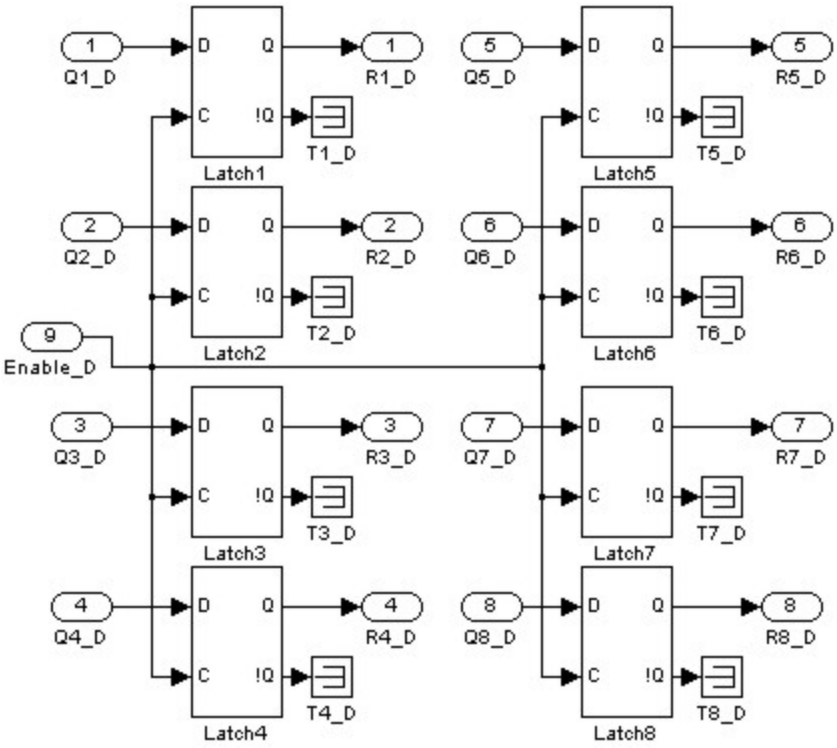

Figure 17. Subsystem that represents internal buffers converter AD7524, being formed by components Terminator and Latch output (the Q_b output from each latch is not used in this work).

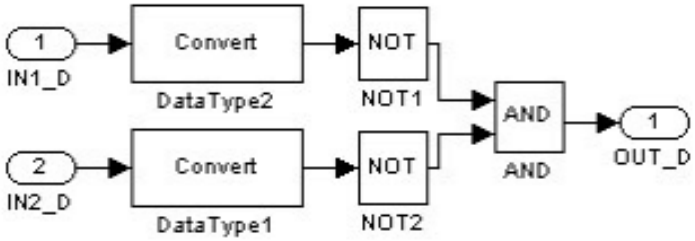

Figure 18. Subsystem that represents the logic controller of AD7524 and formed only by using AND and NOT gates.

http://dx.doi.org/10.5772/57575

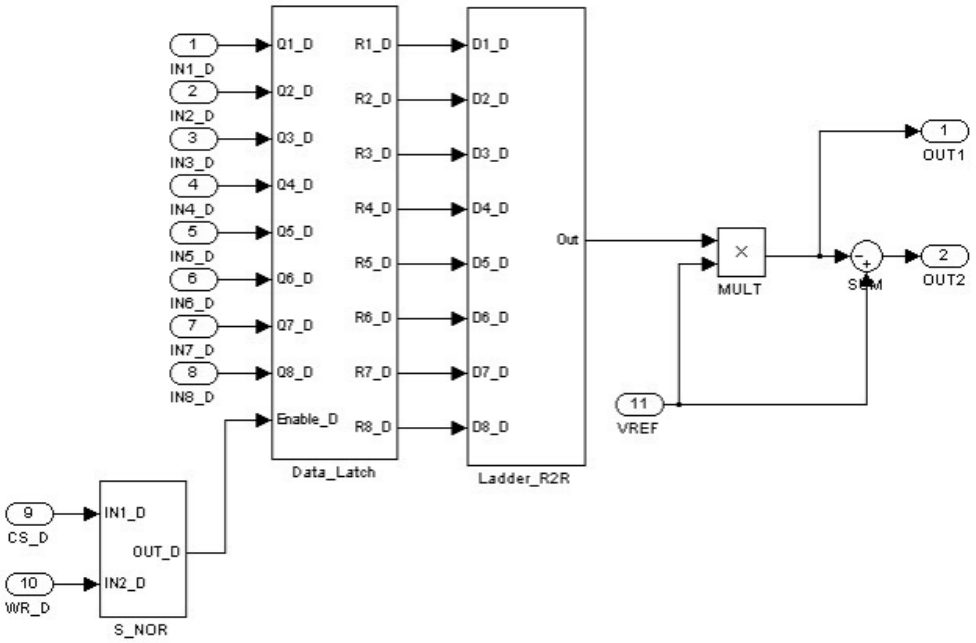

Figure 19. Complete system AD7524 with all subsystems in Figure 16, 17 and 18.

According to the approach described in [40], we created two identical blocks to represent the R/2R ladder, one for the DAC A and one for the DAC B. Similar to the R/2R ladder subsystem of Figure 16, the subsystem of Figure 16 was used twice in the complete system, i.e. one for each block of the AD7528 internal converter (DAC A and DAC B).

Figure 20 represents the logic controller responsible for the selection of which internal drive will be chosen and if converter is in hold or write mode representing behaviour described in Table 2. This subsystem is basically formed by simple logic gates available in toolbox of Simulink as basic primitive [38].

Figure 21 shows the complete AD7528 converter model created in MATLAB / Simulink. In each converter, there are two separate outputs, first one is analog output signal generated by the converter and second one is the difference between analog output and reference voltage. Based on [38] and in order to generate a sine wave signal, MATLAB was able to send the signal to each bit according to its magnitude of the pins of the AD7528 with the "From Workspace" component. Subsequently, the outputs were captured and sent back to MATLAB with the "To Workspace" component to generate the graphic simulation. The reference voltage and the pin configuration for each internal converter were created using "Constant" components (pins DACA/DACB, CS_B and WR_B).

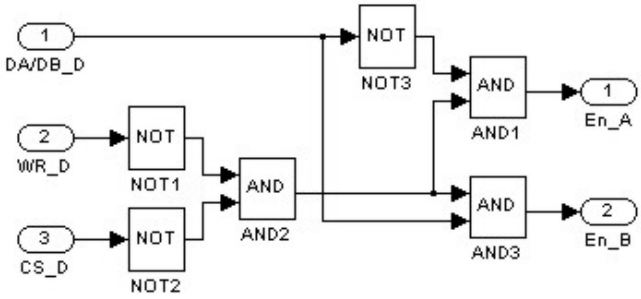

Figure 20. Subsystem that represents logic controller of AD7528 and formed only by using AND and NOT gates.

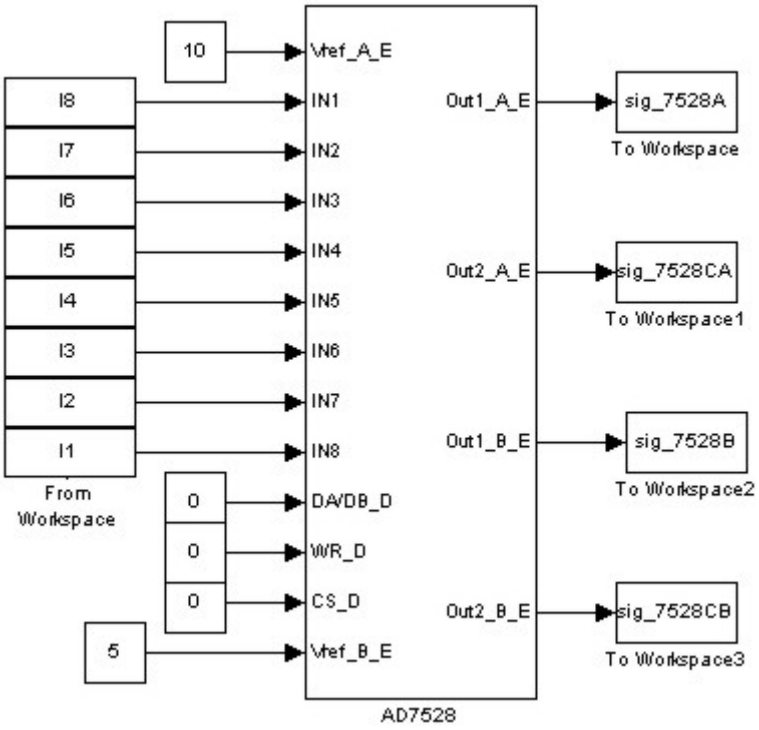

Figure 21. Complete model of AD7528 converter in Simulink with components From Workspace that receive digital signals from MATLAB and sends output back to MATLAB after simulation through To Workspace component.

The DACA/DACB pin could have been used to carry out switching between two converters internally creating two different waveforms from one input. However, as the converters are exactly the same, the kind of simulation does not alter the analysis of signal generated.

http://dx.doi.org/10.5772/57575

5.4. Simulation results of DAC and MS²SV conversion results

After construction and simulation in MATLAB / Simulink, the MS²SV translates model to the SystemVision environment. The hierarchy structure of model in Simulink was maintained in VHDL-AMS codes and MS²SV generated the same configuration of components. The Listing 5 is part of code generated by MS²SV using ladder R/2R in Figure 16. The remains VHDL-AMS description is listed in Appendix A.

```
1.  library IEEE;
2.  use ieee.std_logic_1164.all;
3.  use ieee.electrical_systems.all;
4.  library EduLib;
5.  use WORK.all;
6.  entity Ladder_R2R is
7.  port (
8.  signal D1_D: in std_logic;
9.  signal D2_D: in std_logic;
10. signal D3_D: in std_logic;
11. signal D4_D: in std_logic;
12. signal D5_D: in std_logic;
13. signal D6_D: in std_logic;
14. signal D7_D: in std_logic;
15. signal D8_D: in std_logic;
16. terminal A1: electrical
17. );
18. end entity Ladder_R2R;
19. architecture arch_Ladder_R2R of Ladder_R2R is
20. terminal \1$N0\: electrical;
21. terminal \1$N1\: electrical;
22. ...
23. begin
24. DataType: entity EDULIB.D2A_BIT(IDEAL)
25. generic map ( VHIGH => 1.0,
26. VLOW => 0.0 )
27. port map (
28. D => D1_D,
29. A => \1$N0\
30. );
31. ...
32. E_Gain: entity EDULIB.E_GAIN(BEHAVIORAL)
33. generic map ( K => 0.5 )
34. port map (
35. INPUT => \1$N0\,
36. OUTPUT => \1$N8\
37. );
38. ...
39. E_Sum: entity EDULIB.E_SUM
40. port map (
41. IN1 => \1$N8\,
42. IN2 => \1$N9\,
43. OUTPUT => \1$N16\
44. );
45. ...
46. end architecture arch_Ladder_R2R;
```

Listing 5. Structural VHDL-AMS code generated by MS²SV.

This VHDL-AMS code in Listing 5 has a structural abstraction. It represent the linkage between the components which it has all DAC system, e.g. lines 34-39 represent the ports for Gain block and the linkage between Sum block in lines 41-46 and Data Type Converter block in lines 26-32. The exactly behavior of components are in internal libraries in SystemVision environment named EDULIB (Educational Library).

To perform the simulation, SystemVision was used the same input used in MATLAB / Simulink. Thus was created an input file with the extension.*dat*. It was created to an additional VHDL-AMS code to read this file and send the signal each bit input to the converters, as listed in Listing 6.

```
1.  use std.textio.all;
2.  library IEEE;
3.  use ieee.std_logic_1164.all;
4.  use IEEE.std_logic_textio.all;
5.  entity lerarqp is
6.  port (relogio: in std_logic;
7.    saida: out std_ulogic_vector (8 downto 1));
8.  end entity lerarqp;
9.  architecture behavior of lerarqp is
10. begin
11. testing: process
12.  file f_in: text open READ_MODE is "input.dat";
13.  variable L: LINE;
14.  variable L_BIT: std_ulogic_vector (8 downto 1);
15. begin
16.  while not endfile(f_in) loop
17.    readline (f_in,L);
18.    hread (L,L_BIT);
19.    wait until (relogio'event) and (relogio='1');
20.    saida <= L_BIT;
21.  end loop;
22. end process testing;
23. end architecture behavior;
```

Listing 6. Behavioral VHDL code to read the input simulation file.

In Listing 6 was needed to use a library in order to manipulate files (line 4). It was used for a vector to represent the output which was linked to DAC (line 8). After it was built, a process to read the file and put it into output in lines 13-24 is used.

It was used to save the simulation results in SystemVision in another file with the extension.*dat* so that was possible to analysis the signal generated also in MATLAB, as listed Listing 7.

```
1.  use std.textio.all;
2.  library IEEE;
3.  use IEEE.std_logic_1164.all;
4.  use IEEE.std_logic_textio.all;
5.  entity writewaveform is
6.  port (
7.  signal relogio: in std_logic;
8.  entrada: in real);
9.  end entity writewaveform;
10. architecture behavior of writewaveform is
11. file f_out: text open WRITE_MODE is "C:\Mentor_Projects\DAC08_Test\hdl\output.dat";
12. --quantity escreve_sinal across entrada to ref;
13. begin
14. writying: process
15. variable L: LINE;
16. begin
17.    write(L,REAL'(entrada));
18.    writeline(f_out,L);
19.    wait until (relogio'event) and (relogio='1');
20. end process writying;
21. end architecture behavior;
```

Listing 7. Behavioral VHDL code to write the output simulation file.

Listing 7 follow the same logic in Listing 6, using the library to manipulate file (line 4), a real signal to represent the input which was linked to DAC (line 9) and a process to read from input and write the file in lines 16-22. By space limits, the remains VHDL-AMS description for AD7524 and AD7528 are listed in Appendix B and C, respectively.

For each case study was used a different reference voltage in simulation: 10V, 8V, 6V and 4V, for DAC08, AD7524, AD7528 A and AD7528 B, respectively. Figure 22 shows the simulation

http://dx.doi.org/10.5772/57575

results for 1 second. The different voltages were used to create a better view waveform in Figure 21. However in the SystemVision environment, only 10V was used for all converters and only one waveform is plotted in Figure 21 as all converters presented the same simulation results. In MATLAB / Simulink, the AD7524, AD7528 A and AD7528 B presented noise in output because the delays in flip-flop of type latch.

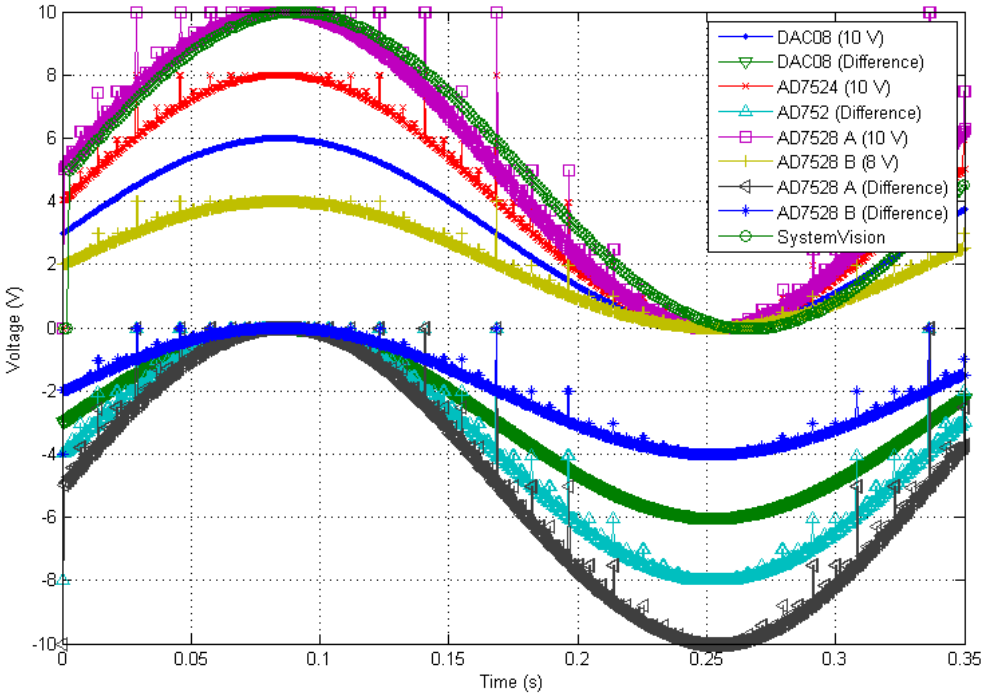

Figure 22. Result of simulation of all case studies in Simulink and signal from SystemVision in simulation total time of 1 second.

Another difference noted in Figure 22 is low simulation time in SystemVision environment compared with simulation MATLAB / Simulink. This was due to the way in which VHDL-AMS sees the frequency inside the SystemVision environment which is different to MATLAB / Simulink.

5.5. Analysis of simulation in MATLAB / Simulink and VHDL-AMS

To verify the quality of signal generated, it was made an analysis of signal obtained through simulation. This analysis was performed through power spectral analysis of the signal obtained from discrete Fourier transform (DFT). Figure 23 illustrates amplitude spectrum of $y(t)$ for all case studies, in MATLAB / Simulink and VHDL-AMS. The graph was generated using the discrete Fourier transform described as [41]:

$$X(k) = \sum_{j=1}^{N} x(j)\omega_N^{(j-1)(k-1)} \tag{7}$$

$$x(j) = (1/N)\sum_{k=1}^{N} X(k)\omega_B^{-(j-1)(k-1)} \tag{8}$$

$$\omega_N = e^{(-2\pi i)/N} \tag{9}$$

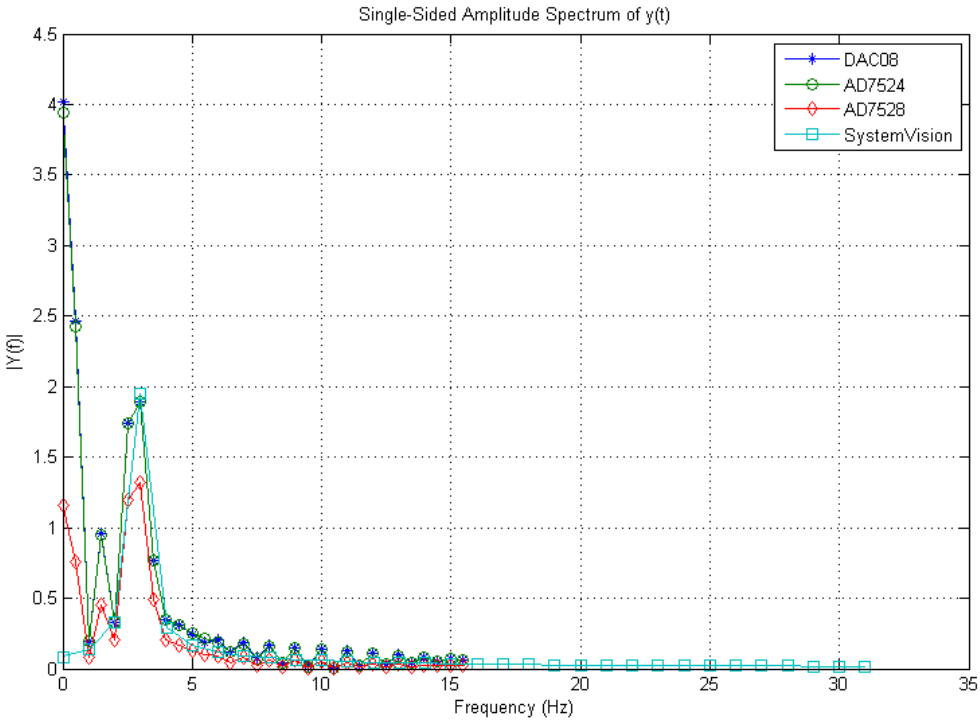

Figure 23. Graph of spectral power all output signals in Simulink and SystemVision with peak in frequency at 2.9297 Hz.

The DC (Direct Current) level of the signal was extracted through shifting the signal along the *y* axis at zero. This resulted in a sine wave signal ranging from-5 to 5 for 10V,-4 to 4 for 8V,-3 to 4 for 6V and-2 to 2 for 4V.

The process of conversion digital to analog represents an interpolation between values of samples provided as inputs, for 20001 points. In this case studies are used linear interpolation,

http://dx.doi.org/10.5772/57575

which points of adjacent samples are connected by a straight line. The system output $x_r(t)$ is given by:

$$x_r(t) = \sum_{n=-\infty}^{+\infty} x(nT)h(t-nT) \tag{10}$$

At the same time, the system has unity gain and linear phase. Systems with this frequency response features produce an output that is a shift in the time of input [41]. Importantly, this is a pseudo-conversion to analog, as the conversion happens only while V_{ref} is constant.

In Figure 23, it is possible to note difference in AD7524, AD7528 A and AD7528 B because of noise in signal. The signal from DAC08 and AD7524 presented a distortion in sine wave showed in Figure 23 too. At the end, the most perfect signal presented by amplitude spectral was the signal from SystemVision environment.

6. Generation code methodology of BD²XML

The computational tool, called BD²XML, is able to read a file block diagram in MATLAB / Simulink and generate a corresponding textual description in XML [42]. A key point of the working methodology is to use a configuration file that allows the user to identify MATLAB / Simulink block to use. This file contains all the blocks readable by the BD²XML and the characteristics that are relevant to operation of the blocks. All this information can be extracted by the user from their own MATLAB / Simulink file through the block properties. The file has tokens that are the key point of interpretation. For example, the token **&** is responsible for the identification of a new block, which has characteristics between the tokens **{** and **}**. Since the token # works as a comment in regular language, i.e., every line that has the # is disregarded. If the token # is used before the token **&** all block is disregarded.

Figure 24 illustrates the functional diagram of BD²XML. Initially begins the reading of.*mdl* file in memory. Next is read the configuration file. All of blocks are loaded in memory to compare with the blocks in.*mdl* file. Only afterwards begins the reading of the.*mdl* file. Relevant information is inside.*mdl* file, such as the type of block, values inside the block, linkage between the blocks and some other information is important to create a netlist of circuit. From this point, it is possible to create an XML file with the circuit that can used, and reused in many contexts.

BD²XML was developed according to class diagram showed in Figure 25. The classes responsible for creating a logical structure in memory are inherited by classes responsible for read the configuration file and the block diagram file of the MATLAB / Simulink. The **misc** class with routines common to other classes is also inherited by other classes.

Others instances of classes are composition. This facilitates the creation of threads structure in future to speeding of codes performance for generations of multiple objects codes simultaneously. The designer starts the process specifying the diagram which is the target of BD²XML.

At first, the configuration file is read into a structure and then the block diagram of the MATLAB / Simulink is read into another structure. A comparison is done block by block to check whether the target block is identifiable. Every block of the configuration file has its own characteristics and the default values for these characteristics. In the case of not being identified in some of the characteristics relevant target block are considered the default values of the configuration file to generate the XML. If any block is not identifiable, XML is generated in the same way discarding that particular block.

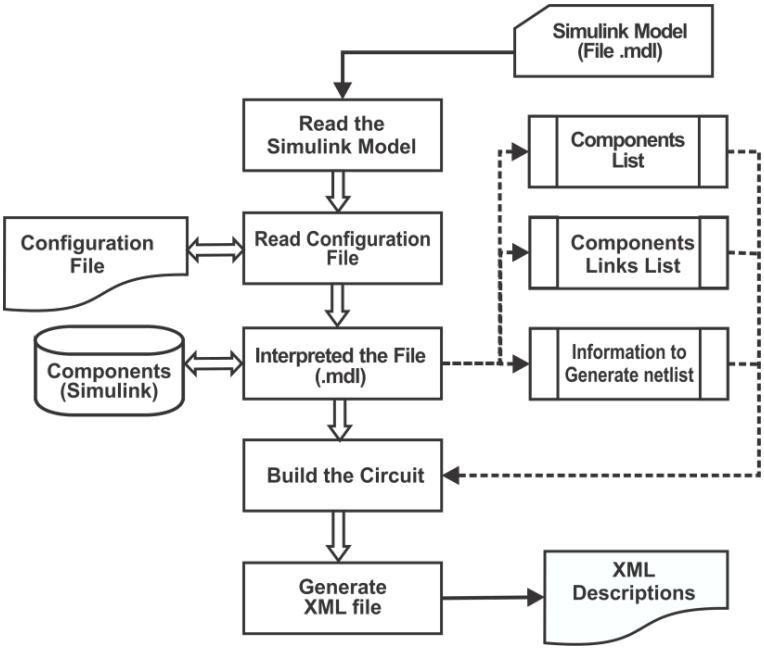

Figure 24. Class diagram of the BD2XML tool.

6.1. Conversion results using AD7528

To demonstrate the BD²XML, we used the Ladder R/2R case study in Figure 16. Listing 8 is shown a partial description in XML generated by the BD²XML from the Ladder R/2R.

The tags **<Diagram>** and **</Diagram>** (lines 2-57) identifying the block diagram. The blocks are identified by tags **<BasicBlock type="..." name="...">** and **</BasicBlock>**, being specified block type and the name used in MATLAB / Simulink, for example, the gain block (lines 30-33).

http://dx.doi.org/10.5772/57575

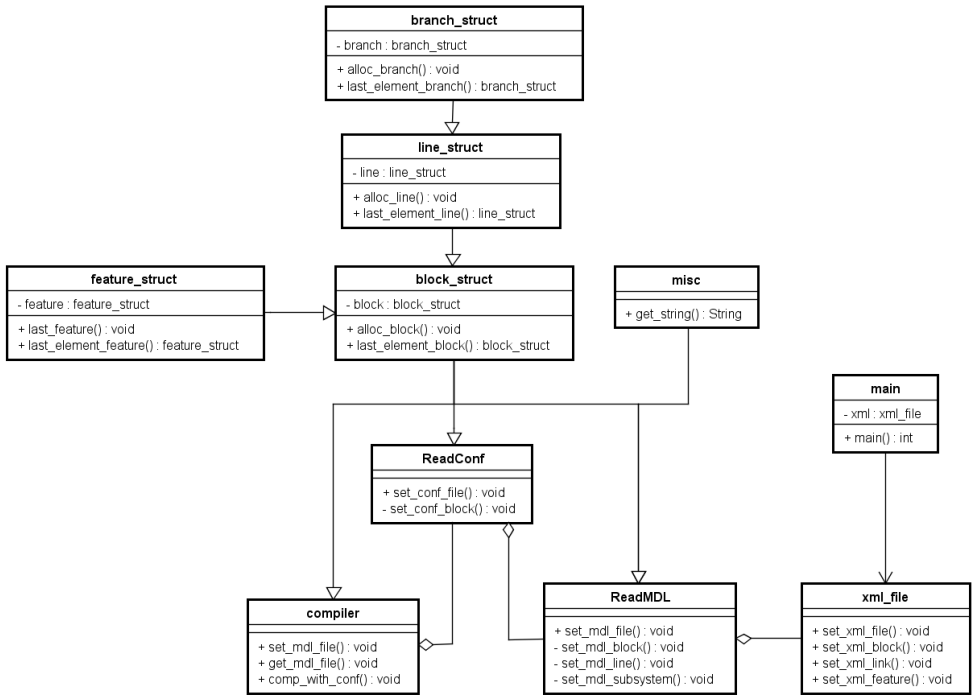

Figure 25. Class diagram of the BD²XML tool.

The data blocks are specified by the tag **<data type="..." value="..." />**. Even in the case of gain, the first tag **data** specifies the kind of signal is used by the block (line 31), e.g., analog. The next type of data (line 32) specifies the gain value that block has.

In the configuration file, the blocks of gain are represented in accordance with the Listing 9 (lines 11-14). The block gain has no specification of input and output ports, because by default a gain always has an input and an output.

The connections between the blocks are made through the tags **<Link>** and **</Link>** (Listing 8, lines 15-20). Inside these tags are two tags named **<OutputPort>** and **</OutputPort>**, that specify the output ports of the block owned by **as="source"** and indicating which is the block output **id="..."**.

The property **id** is understood as the property **name** of the tag **BasicBlock**. Similarly, the entry are specified by the tags **<InputPort>** and **</InputPort>**, and property **as="target"**.

If there are subsystems in the block diagram, they are identified by the tags **<SubSystem name="...">** and **</SubSystem>**, according to the Listing 8 (lines 7-22, 23-40 and 51-55). All configuration of the subsystem is between two tags and their description is the same as already explained.

```
1.  <?xml version="1.0" encoding="ISO-8859-1"?>
2.  <Diagram>
3.  <InBlock name="in1_d">
4.  <data type="mixed"/>
5.  </InBlock>
6.  ...
7.  <SubSystem name="datalatch_d">
8.  <data type="mixed"/>
9.  <data type="ports" value="9, 8"/>
10. ...
11. <OutBlock name="r7_d">
12. <data type="mixed"/>
13. </OutBlock>
14. ...
15. <Link>
16. <OutputPort as="source" id="q1_d"/>
17. </OutputPort>
18. <InputPort as="target" id="latch1"/>
19. </InputPort>
20. </Link>
21. ...
22. </SubSystem>
23. <SubSystem name="ladderr2r">
24. <data type="mixed"/>
25. <data type="ports" value="8, 1"/>
26. <InBlock name="d1_d">
27. <data type="mixed"/>
28. </InBlock>
29. ...
30. <BasicBlock type="gain" name="gain">
31. <data type="analog"/>
32. <data type="gain" value="0.5"/>
33. </BasicBlock>
34. ...
35. <BasicBlock type="sum" name="sum1">
36. <data type="analog"/>
37. <data type="ports" value="2, 1"/>
38. </BasicBlock>
39. ...
40. </SubSystem>
41. <BasicBlock type="product" name="mult">
42. <data type="analog"/>
43. <data type="ports" value="2, 1"/>
44. </BasicBlock>
45. <SubSystem name="snor_d">
46. <data type="mixed"/>
47. <data type="ports" value="2, 1"/>
48. ...
49. <BasicBlock type="logic" name="and1">
50. <data type="digital"/>
51. <data type="ports" value="2, 1"/>
52. <data type="operator" value="and"/>
53. </BasicBlock>
54. ...
55. </SubSystem>
56. ...
57. </Diagram>
```

Listing 8. XML description of Ladder R/2R.

Tags to indicate input and output ports of the block diagram are used most commonly in subsystems and they are in accordance with Listing 8 (lines 3-5, 11-13 and 26-28). Tags to represent input ports are specified by **<InBlock name="...">** and **</InBlock>** (lines 3-5).

Similarly, the output ports are specified by the tags **<OutBlock name="...">** and **</OutBlock>** (Listing 8, lines 11-13). In both the tag property **name** identify the block. Also in either case the tag **<data... />** (Listing 8, lines 4 and 8) are of mixed type, since both blocks can be an input or an output of digital and analog blocks.

All the representation was generated with perfection in one complete file containing all the blocks and subsystems. In this case study AD7528, there are five subsystems representing: the block **Ladder R/2R A**, **Ladder R/2R B**, **Datalatch A**, **Datalatch B** and **Control Logic**.

http://dx.doi.org/10.5772/57575

```
1.  &inport {
2.  mixed;
3.  }
4.  &outport {
5.  mixed;
6.  }
7.  &sum {
8.  analog;
9.  ports   "2,1";
10. }
11. &gain {
12. analog;
13. gain    "1";
14. }
15. &logic {
16. digital;
17. ports   "2,1";
18. operator        "and";
19. }
20. &toworkspace {
21. #
22. }
23. &fromworkspace {
24. #
25. }
26. &subsystem {
27. mixed;
28. ports   "1,1";
29. }
```

Listing 9. Representation of the blocks in the configuration file used by BD²XML.

7. Generation code methodology of SF²XML

The in this section, the tool called SF²XML is presented which is capable to capture relevant information in the Stateflow environment and generate a corresponding description in XML [43]. The resulting file conversion is also in accordance with international standards of default file in case the SCXML proposed by W3C [4].

The SF²XML has four classes of objects, an object that makes up the logical structure of storage memory at runtime, an object of reading the file MATLAB / Simulink and extraction of relevant information, an object to generate the syntactic and semantic file XML and other objects to make interfacing with the user. The class diagram to SF²XML is shown in Figure 26. In this figure, it is possible to see that the only difference between SF²XML and SF²HDL is the highlighted **xml_file** object. Similarly, the functional diagram in Figure 5, is the same to SF²XM apart from the files generated.

SF²XML is also able to identify sub-finite state machine, so it has recursive methods for hierarchical finite state machine clustering automatically, which generates a certain economy of code and maintenance. There is this feature inside SF²HDL too, but a hierarchical FSM can generate instable behavioral in a same file. It could better if SF²HDL would generate different files for different finite state machines. SF²XML has the follow algorithm, which can be understood for SF²HDL:

1. Read the MATLAB / Simulink file and identify the FSM;
2. Store the state or transitions and its information;
    a. If the state has a sub-FSM store the first transition in this sub-FSM, otherwise go to step 2(b).

b. Verify which type this FSM is (Mealy or Moore);

3. Start write XML file;

a. a. Seeking relationship between states of the same hierarchical level and which state is the first in highest level;

b. Write the state tags, the transition related to this states;

c. If there is a sub-FSM decrease the level hierarchical and go to step 3(a), otherwise go to step 3(d);

d. While there is a state such that is associated with a transition go to step 3(a).

4. Return the XML file.

7.1. Conversion results of HDB3 line code

The tool was able to accurately generate an XML representation corresponding to the diagram shown in Figure 3. Every XML description is according to SCXML specification. Listing 10 below displays the entire description in XML generated by SF²XML.

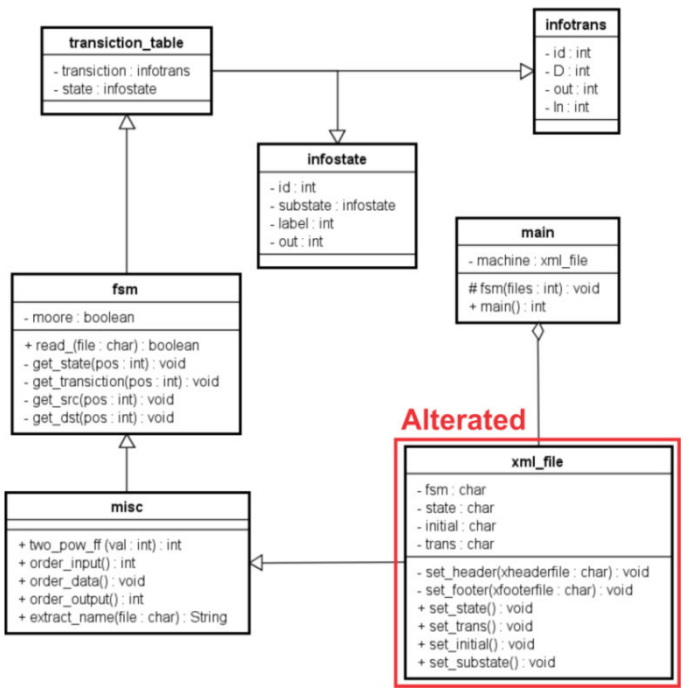

Figure 26. Class diagram of the SF²XML tool.

http://dx.doi.org/10.5772/57575

```
1.  <?xml version="1.0" encoding="ISO-8859-1"?>
2.  <scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml" initialstate="0">
3.  <state id="0">
4.  <transition event="INPUT=1">
5.  <target next="1" />
6.  <assign expr="OUTPUT=1" />
7.  </transition>
8.  <transition event="INPUT=0">
9.  <target next="0" />
10. <assign expr="OUTPUT=0" />
11. </transition>
12. </state>
13. <state id="1">
14. <transition event="INPUT=0">
15. <target next="2" />
16. <assign expr="OUTPUT=0" />
17. </transition>
18. </state>
19. ...
20. </scxml>
```

Listing 10. XML code representing the finite state machine of HDB3.

It is important to highlight that states assumed a decimal representation continuously, regardless of whether the state is within another state or not.

Line 1 of Listing 10 specifies that it is an XML file, the XML version and encoding used. The tags of lines 2 and 20, specify the start and end of the finite state machine. In line 2 is also specified the version of SCXML used, the name for the XML document (xmlns) and the initial state of the finite state machine (**initialstate="0"**). The tag **<state id="...">** and **</state>** specifies the state and its identifier (lines 3, 12, 13, 18). The tags **<transition event="...">** and **</transition>** to specify a transition that will occur according to a particular event, e.g. **" INPUT=1"** (line 4). If the transition occurs, the tag **<target next="..." />** specifies the next state of the FSM according to its identifier. The tag **<assign expr="OUTPUT=..." />** assigns a new value to the output attribute **expr**. If there are hierarchically nested state, states are chained between the tags **<state id="...">** and **</state>**. However, the initial state is specified by internal tags **<initial>** and **</initial>** within a transition indicating to the initial state.

8. Conclusion

The SF²HDL program makes the translation of finite state machine, but it does not perform circuit optimization. Hence, it also needs a translation into a file that serves as input to TABELA program that is able to perform this work.

The SF²HDL program does not make optimal state assignment to the state machine as well. Thus, it is necessary to use another tool to do that task. In this way circuit synthesis can be really optimize using all the potentialities of developed tools. Future work will extend this tool to also realize the translation of finite state machines for the structural VHDL and Verilog code.

The use of CAD tools and a top-down methodology is a reality that is present in the most electronic circuit design projects, and the development of new computational tools to aid in

project implementation is highly desirable, because the range of CAD tools used introduces incompatibilities that can be time consuming and costly in translating between them. Hence, there is the need for suitable tools to be able to translate models between different representations.

The SF^{2}HDL tool was tested in several cases of finite state machines and all have been simulated and synthesized in Quartus II environment and Cyclone II FPGA model EP2C20F484C7, respectively.

For the case study of HDB3 code described in this chapter, SF^{2}HDL program was very effective for translation code modeled on Stateflow, a computational low cost, being accurate in VHDL generation code and the input file for TABELA program corresponding with another programs.

Thus, this research generated a tool that allows to the designers to model a finite state machine at high level of abstraction and obtain a corresponding VHDL model which can be synthesized on synthesis tools available commercially. It is also important to say this new translation tools interact with several other tools developed by research group with TABELA program, creating this way a digital synthesis environment.

The MS^{2}SV tool proved convenient to use as it allows for the creation of multiple libraries for different projects and also recycle the libraries for use in others projects. This allows a significant savings of time and hence costs.

By analysing power spectral of a discrete-time signal, one can say that the output signals in MATLAB / Simulink and VHDL-AMS are almost identical. This fact allows us to change our vision about the abstraction levels and details in modelling. Even in the same environment (MATLAB / Simulink) we have difference in spectral signal. If the system be simulating in another environment, we will also have different simulations.

The conversion to XML (using BD^{2}XML) is convenient because the portability of the language allows a wide applicability of the proposed methodology, e.g., documentation, level synthesis of hardware or software, web publishing, use of XML simulators and as object code for future work. This is the same when using SF^{2}XML, which it use a standardized description in SCXML.

The use of the intermediate code (like XML) is very important since it can be used in frameworks generation and optimization of other types of coding. As an example, hardware description languages like VHDL-AMS, Verilog-AMS, SystemC, etc. Or even reprogrammable devices that have analog and digital blocks and PSoC (Programmable System-on-Chip).

This point highlights the importance to explore more alternatives and standard methodologies of electronic circuit design. Nevertheless, our methodology and tools need further investigation and analysis to consider the need for the development of larger and more complex systems.

http://dx.doi.org/10.5772/57575

Appendix A– DAC08 converter description in structural VHDL-AMS

The complete VHDL-AMS code is listed below for DAC08 case.

```
1.    library IEEE;
2.    use ieee.std_logic_1164.all;
3.    use ieee.electrical_systems.all;
4.     library EduLib;
5.     use WORK.all;
6.     entity DAC08 is
7.     end entity DAC08;
8.    architecture arch_DAC08 of DAC08 is
9.        terminal \1$N39\: electrical;
10.       terminal \1$N41\: electrical;
11.       signal DIn1_D: std_logic;
12.       signal DIn2_D: std_logic;
13.       signal DIn3_D: std_logic;
14.       signal DIn4_D: std_logic;
15.       signal DIn5_D: std_logic;
16.       signal DIn6_D: std_logic;
17.       signal DIn7_D: std_logic;
18.       signal DIn8_D: std_logic;
19.       terminal Vref: electrical;
20.       terminal Out1: electrical;
21.    begin
22.       Ladder_R2R: entity WORK.Ladder_R2R(arch_Ladder_R2R)
23.           port map (
24.               D1_D => DIn1_D,
25.               D2_D => DIn2_D,
26.               D3_D => DIn3_D,
27.               D4_D => DIn4_D,
28.               D5_D => DIn5_D,
29.               D6_D => DIn6_D,
30.               D7_D => DIn7_D,
31.               D8_D => DIn8_D,
32.               A1 => \1$N39\    );
33.       E_MULT: entity EDULIB.E_MULT
34.           port map (
35.               IN1 => \1$N39\,
36.               IN2 => Vref,
37.               OUTPUT => Out1    );
38.       V_VREF: entity EDULIB.V_CONSTANT(IDEAL)
39.         generic map ( LEVEL => 10.0 )
40.         port map ( POS => Vref,
41.               NEG => ELECTRICAL_REF);
42.    end architecture arch_DAC08;
```

Appendix B– AD7524 converter descriptions in structural VHDL-AMS

The complete VHDL-AMS code is listed below for Data_Latch subsystem, SNOR subsystem and AD7524 converter.

- Data_Latch

```
1.    library IEEE;
2.    use ieee.std_logic_1164.all;
3.    use ieee.electrical_systems.all;
4.     library EduLib;
5.     use WORK.all;
6.     entity DataLatch_D is
7.       port (
8.            signal Q1_D: in std_logic;
9.        signal Q2_D: in std_logic;
10.        signal Q3_D: in std_logic;
11.        signal Q4_D: in std_logic;
12.        signal Q5_D: in std_logic;
13.        signal Q6_D: in std_logic;
14.        signal Q7_D: in std_logic;
15.        signal Q8_D: in std_logic;
16.        signal Enable_D: in std_logic;
17.        signal R1_D: out std_logic;
18.        signal R2_D: out std_logic;
19.        signal R3_D: out std_logic;
20.        signal R4_D: out std_logic;
21.        signal R5_D: out std_logic;
22.        signal R6_D: out std_logic;
23.        signal R7_D: out std_logic;
24.        signal R8_D: out std_logic );
25.    end entity DataLatch_D;
26.    architecture arch_DataLatch_D of DataLatch_D is
27.        signal \1$N2\: std_logic;
28.        signal \1$N5\: std_logic;
29.        signal \1$N8\: std_logic;
30.        signal \1$N11\: std_logic;
31.        signal \1$N23\: std_logic;
32.        signal \1$N16\: std_logic;
33.        signal \1$N19\: std_logic;
34.        signal \1$N22\: std_logic;
35.    begin
36.        DLatch: entity WORK.DLatch(arch_DLatch)
37.            port map (
38.                D => Q1_D,
39.                CLK => Enable_D,
40.                Q => R1_D,
41.                QN => \1$N2\);
42.        DLatch1: entity WORK.DLatch(arch_DLatch)
43.            port map (
44.                D => Q2_D,
```

http://dx.doi.org/10.5772/57575

```
45.                CLK => Enable_D,
46.                Q => R2_D,
47.                QN => \1$N5\);
48.        DLatch2: entity WORK.DLatch(arch_DLatch)
49.            port map (
50.                D => Q3_D,
51.                CLK => Enable_D,
52.                Q => R3_D,
53.                QN => \1$N8\);
54.        DLatch3: entity WORK.DLatch(arch_DLatch)
55.            port map (
56.                D => Q4_D,
57.                CLK => Enable_D,
58.                Q => R4_D,
59.                QN => \1$N11\    );
60.        DLatch4: entity WORK.DLatch(arch_DLatch)
61.            port map (
62.                D => Q5_D,
63.                CLK => Enable_D,
64.                Q => R5_D,
65.                QN => \1$N23\    );
66.        DLatch5: entity WORK.DLatch(arch_DLatch)
67.            port map (
68.                D => Q6_D,
69.                CLK => Enable_D,
70.                Q => R6_D,
71.                QN => \1$N16\);
72.        DLatch6: entity WORK.DLatch(arch_DLatch)
73.            port map (
74.                D => Q7_D,
75.                CLK => Enable_D,
76.                Q => R7_D,
77.                QN => \1$N19\    );
78.        DLatch7: entity WORK.DLatch(arch_DLatch)
79.            port map (
80.                D => Q8_D,
81.                CLK => Enable_D,
82.                Q => R8_D,
83.                QN => \1$N22\    );
84.    end architecture arch_DataLatch_D;
```

- D Latch

```
1.    LIBRARY IEEE;
2.    USE ieee.std_logic_1164.all;
3.    USE ieee.electrical_systems.all;
4.     LIBRARY edulib;
5.     USE work.all;
6.     entity DLatch is
7.      port (
8.        signal D: in std_logic;
9.      signal CLK: in std_logic;
```

```
10.      signal Q: out std_logic;
11.      signal QN: out std_logic );
12.    end entity DLatch;
13.    architecture arch_DLatch of DLatch is
14.    begin
15.    process(D,CLK)
16.    begin
17.     if clk = '1' then
18.      Q <= D;
19.      QN <= not D;
20.     end if;
21.    end process;
22.    end architecture arch_DLatch;
```

- SNOR

```
1.    library IEEE;
2.    use ieee.std_logic_1164.all;
3.    use ieee.electrical_systems.all;
4.    library EduLib;
5.    use WORK.all;
6.    entity SNOR_D is
7.    port (
8.        signal IN1_D: in std_logic;
9.        signal IN2_D: in std_logic;
10.       signal OUT_D: out std_logic );
11.   end entity SNOR_D;
12.   architecture arch_SNOR_D of SNOR_D is
13.       signal \1$N63\: std_logic;
14.       signal \1$N64\: std_logic;
15.   begin
16.       G_AND1: entity EDULIB.AND2
17.           port map (
18.               IN1 => \1$N63\,
19.               IN2 => \1$N64\,
20.               OUTPUT => OUT_D);
21.       G_NOT1: entity EDULIB.INVERTER
22.           port map (
23.               INPUT => IN1_D,
24.               OUTPUT => \1$N63\);
25.       G_NOT2: entity EDULIB.INVERTER
26.           port map (
27.               INPUT => IN2_D,
28.               OUTPUT => \1$N64\);
29.   end architecture arch_SNOR_D;
```

- AD7524

```
1.    library IEEE;
2.    use ieee.std_logic_1164.all;
3.    use ieee.electrical_systems.all;
4.    use ieee.radiant_systems.all;
```

http://dx.doi.org/10.5772/57575

```
5.    library EduLib;
6.    use WORK.all;
7.    entity AD7524 is
8.    end entity AD7524;
9.    architecture arch_AD7524 of AD7524 is
10.     signal IN1_D: std_logic;
11.        signal IN2_D: std_logic;
12.        signal IN3_D: std_logic;
13.        signal IN4_D: std_logic;
14.        signal IN5_D: std_logic;
15.        signal IN6_D: std_logic;
16.        signal IN7_D: std_logic;
17.        signal IN8_D: std_logic;
18.        signal CS_D: std_logic;
19.        signal WR_D: std_logic;
20.        terminal VREF: electrical;
21.        terminal OUT1: electrical;
22.        terminal OUT2: electrical;
23.        signal \1$N68\: std_logic;
24.        signal \1$N69\: std_logic;
25.        signal \1$N70\: std_logic;
26.        signal \1$N71\: std_logic;
27.        signal \1$N72\: std_logic;
28.        signal \1$N73\: std_logic;
29.        signal \1$N74\: std_logic;
30.        signal \1$N75\: std_logic;
31.        terminal \1$N78\: electrical;
32.        terminal WR: electrical;
33.        terminal CS: electrical;
34.        signal \1$N92\: std_logic;
35.    begin
36.        DataLatch_D: entity WORK.DataLatch_D(arch_DataLatch_D)
37.            port map (
38.                Q1_D => IN1_D,
39.                Q2_D => IN2_D,
40.                Q3_D => IN3_D,
41.                Q4_D => IN4_D,
42.                Q5_D => IN5_D,
43.                Q6_D => IN6_D,
44.                Q7_D => IN7_D,
45.                Q8_D => IN8_D,
46.                Enable_D => \1$N92\,
47.                R1_D => \1$N68\,
48.                R2_D => \1$N69\,
49.                R3_D => \1$N70\,
50.                R4_D => \1$N71\,
51.                R5_D => \1$N72\,
52.                R6_D => \1$N73\,
53.                R7_D => \1$N74\,
54.                R8_D => \1$N75\);
55.        LadderR2R: entity WORK.LadderR2R(arch_LadderR2R)
56.            port map (
```

```
57.                D1_D => \1$N68\,
58.                D2_D => \1$N69\,
59.                D3_D => \1$N70\,
60.                D4_D => \1$N71\,
61.                D5_D => \1$N72\,
62.                D6_D => \1$N73\,
63.                D7_D => \1$N74\,
64.                D8_D => \1$N75\,
65.                Out1 => \1$N78\);
66.        E_MULT: entity EDULIB.E_MULT
67.            port map (
68.                IN1 => \1$N78\,
69.                IN2 => VREF,
70.                OUTPUT => OUT1    );
71.        SNOR_D: entity WORK.SNOR_D(arch_SNOR_D)
72.            port map (
73.                IN1_D => CS_D,
74.                IN2_D => WR_D,
75.                OUT_D => \1$N92\);
76.        E_SUM: entity EDULIB.E_SUM
77.            port map (
78.                IN1 => \1$N78\,
79.                IN2 => VREF,
80.                OUTPUT => OUT2    );
81.        V_CONT1: entity EDULIB.V_CONSTANT(IDEAL)
82.          generic map ( LEVEL => 0.0 )
83.          port map ( POS => CS,
84.                NEG => ELECTRICAL_REF);
85.
86.        A2D_BIT1: entity EDULIB.A2D_BIT(IDEAL)
87.            --generic map ( LEVEL => 1.0 )
88.            port map (A => CS,
89.                   D => CS_D);
90.        V_CONT2: entity EDULIB.V_CONSTANT(IDEAL)
91.          generic map ( LEVEL => 0.0 )
92.          port map ( POS => WR,
93.                NEG => ELECTRICAL_REF);
94.        A2D_BIT2: entity EDULIB.A2D_BIT(IDEAL)
95.            --generic map ( LEVEL => 1.0 )
96.            port map (A => WR,
97.                   D => WR_D);
98.        V_REF: entity EDULIB.V_CONSTANT(IDEAL)
99.          generic map ( LEVEL => 10.0 )
100.          port map ( POS => VREF,
101.                NEG => ELECTRICAL_REF);
102.    end architecture arch_AD7524;
```

Appendix C– AD7528 converter descriptions in structural VHDL-AMS

The complete VHDL-MAS code is listed for Control_Logic and AD7528 converter.

http://dx.doi.org/10.5772/57575

- Control_Logic

```
1.    library IEEE;
2.    use ieee.std_logic_1164.all;
3.    use ieee.electrical_systems.all;
4.    library EduLib;
5.    use WORK.all;
6.    entity ControlLogic_D is
7.    port (
8.        signal DADB_D: in std_logic;
9.        signal WR_D: in std_logic;
10.        signal CS_D: in std_logic;
11.        signal EnA_D: out std_logic;
12.        signal EnB_D: out std_logic );
13.    end entity ControlLogic_D;
14.    architecture arch_ControlLogic_D of ControlLogic_D is
15.        signal \1$N9\: std_logic;
16.        signal \1$N2\: std_logic;
17.        signal \1$N3\: std_logic;
18.        signal \1$N4\: std_logic;
19.    begin
20.        G_AND1: entity EDULIB.AND2
21.            port map (
22.                IN1 => \1$N2\,
23.                IN2 => \1$N3\,
24.                OUTPUT => \1$N9\);
25.        G_AND2: entity EDULIB.AND2
26.            port map (
27.                IN1 => \1$N4\,
28.                IN2 => \1$N9\,
29.                OUTPUT => EnA_D);
30.        G_AND3: entity EDULIB.AND2
31.            port map (
32.                IN1 => DADB_D,
33.                IN2 => \1$N9\,
34.                OUTPUT => EnB_D);
35.        G_NOT1: entity EDULIB.INVERTER
36.            port map (
37.                INPUT => WR_D,
38.                OUTPUT => \1$N2\);
39.        G_NOT2: entity EDULIB.INVERTER
40.            port map (
41.                INPUT => CS_D,
42.                OUTPUT => \1$N3\);
43.        G_NOT3: entity EDULIB.INVERTER
44.            port map (
45.                INPUT => DADB_D,
46.                OUTPUT => \1$N4\ );
47.    end architecture arch_ControlLogic_D;
```

- AD7528

```
1.    library IEEE;
2.    use ieee.std_logic_1164.all;
3.    use ieee.electrical_systems.all;
4.    library EduLib;
5.    use WORK.all;
6.    entity AD7528 is
7.    end entity AD7528;
8.    architecture arch_AD7528 of AD7528 is
9.     signal IN1_D: std_logic;
10.        signal IN2_D: std_logic;
11.        signal IN3_D: std_logic;
12.        signal IN4_D: std_logic;
13.        signal IN5_D: std_logic;
14.        signal IN6_D: std_logic;
15.        signal IN7_D: std_logic;
16.        signal IN8_D: std_logic;
17.        signal DA_DB_D: std_logic;
18.        signal WR_D: std_logic;
19.        signal CS_D: std_logic;
20.        terminal CS: electrical;
21.        terminal WR: electrical;
22.        terminal DA_DB: electrical;
23.        terminal VrefA: electrical;
24.        terminal VrefB: electrical;
25.        terminal Out1A: electrical;
26.        terminal Out1B: electrical;
27.        signal \1$N204\: std_logic;
28.        signal \1$N205\: std_logic;
29.    begin
30.        ControlLogic_D: entity WORK.ControlLogic_D(arch_ControlLogic_D)
31.            port map (
32.                DADB_D => DA_DB_D,
33.                WR_D => WR_D,
34.                CS_D => CS_D,
35.                EnA_D => \1$N204\,
36.                EnB_D => \1$N205\    );
37.        DAC_A: entity WORK.DAC_A(arch_DAC_A)
38.            port map (
39.                A1_D => IN1_D,
40.                A2_D => IN2_D,
41.                A3_D => IN3_D,
42.                A4_D => IN4_D,
43.                A5_D => IN5_D,
44.                A6_D => IN6_D,
45.                A7_D => IN7_D,
46.                A8_D => IN8_D,
47.                VrefA_I => VrefA,
48.                EnA_D => \1$N204\,
49.                OutA => Out1A    );
50.        DAC_B: entity WORK.DAC_B(arch_DAC_B)
51.            port map (
52.                B1_D => IN1_D,
```

http://dx.doi.org/10.5772/57575

```
53.                B2_D => IN2_D,
54.                B3_D => IN3_D,
55.                B4_D => IN4_D,
56.                B5_D => IN5_D,
57.                B6_D => IN6_D,
58.                B7_D => IN7_D,
59.                B8_D => IN8_D,
60.                VrefB_I => VrefB,
61.                EnB_D => \1$N205\,
62.                OutB => Out1B );
63.        V_CONST1: entity EDULIB.V_CONSTANT(IDEAL)
64.          generic map ( LEVEL => 0.0 )
65.          port map ( POS => CS,
66.                NEG => ELECTRICAL_REF);
67.        A2D_BIT1: entity EDULIB.A2D_BIT(IDEAL)
68.            --generic map ( LEVEL => 1.0 )
69.            port map (A => CS,
70.                   D => CS_D);
71.        V_CONST2: entity EDULIB.V_CONSTANT(IDEAL)
72.          generic map ( LEVEL => 0.0 )
73.          port map ( POS => WR,
74.                NEG => ELECTRICAL_REF);
75.        A2D_BIT2: entity EDULIB.A2D_BIT(IDEAL)
76.            --generic map ( LEVEL => 1.0 )
77.            port map (A => WR,
78.                   D => WR_D);
79.        V_CONST3: entity EDULIB.V_CONSTANT(IDEAL)
80.          generic map ( LEVEL => 0.0 )
81.          port map ( POS => DA_DB,
82.                NEG => ELECTRICAL_REF);
83.        A2D_BIT3: entity EDULIB.A2D_BIT(IDEAL)
84.            --generic map ( THRES => 1.0 )
85.            port map (A => DA_DB,
86.                   D => DA_DB_D);
87.        V_REFA: entity EDULIB.V_CONSTANT(IDEAL)
88.          generic map ( LEVEL => 16.0 )
89.          port map ( POS => VrefA,
90.                NEG => ELECTRICAL_REF);
91.        V_REFB: entity EDULIB.V_CONSTANT(IDEAL)
92.          generic map ( LEVEL => 10.0 )
93.          port map ( POS => VrefB,
94.                NEG => ELECTRICAL_REF
95.    end architecture arch_AD7528;
```

Acknowledgements

The authors would like to thank to National Council for Scientific and Technological Development (CNPq, process: 141744/2010-3 and 309023/2012-2).

Author details

Tiago da Silva Almeida[1], Ian Andrew Grout[2] and Alexandre César Rodrigues da Silva[1]

1 Univ Estadual Paulista, UNESP, Brazil

2 University of Limerik, UL, Ireland

References

[1] Gerstlauer A, Haubelt C, Pimentel A D, Stefanov T P, Gajski D D, Teich J. Electronic System-Level Synthesis Methodologies. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 2009; 28(10)1517-1530.

[2] Gajski D D, Kuhn R H. Introduction new VLSI tools. IEEE Computer 1983; 16(12) 11-14.

[3] Riesgo T, Torroja Y, Torre E. Design methodologies based on hardware description languages. IEEE Transactions on Industrial Electronics 1999; 46(1) 3-12.

[4] Auburn R, Barnett J, Bodell M, Raman T. State chart XML (SCXML): State machine notation for control abstraction 1.0. http://www.w3.org/TR/2005/WD-scxml-20050705/ (accessed 5 September 2013).

[5] Tocci R J, Widmer N S, Moss G L. Digital Systems: Principles and Applications. New Jersey, USA: Pearson; 2008.

[6] Keogh D B. The State Diagram of HDB3. IEEE Transactions on Communications 1984; 32(11) 1222-1224.

[7] Silva A C R. Contribuição a Minimização e Simulação de Circuitos Lógicos. Master degrees. Universidade Estadual de Campinas, Campinas, Brazil; 1989.

[8] Umans C, Villa T, Sangiovanni-Vicentelli A L. Complexity of two-level logic minimization. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 2006; 25(10) 1230-1246.

[9] Su A P. Application of ESL Synthesis on GSM Edge Algorithm for Base Station. In: Asia And South Pacific Design Automation Conference, 2010, Taipei, Taiwan; 2010.

[10] Yuan L et al. An FSM Reengineering Approach to Sequential Circuit Synthesis by State Splitting. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 2008; 27(6) 1159-1164.

[11] Nehme C, Lundqvist K. A Tool For Translating VHDL to Finite State Machines.In: Digital Avionics Systems Onference, 2003, USA; 2003.

[12] Matrosova A, Ostanin S. Self-cheking FSM design with observing only FSM output. In: IEEE On-Line Testing Workshop, 2000, Palma de Mallorca, Spain; 2000.

[13] Xia L, Bell I M, Wilkison A J. Automated model generation algorithm for high-level fault modeling. IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems 2010;29(7) 1140-1145.

[14] Correa I S et al. VHDL Implementation of a Flexible and Synthesizable FFT Processor. IEEE Latin America Transaction, São Paulo 2012; 10(1) 1180-1183.

[15] Kapre N, Dehon A. SPICE2: Spatial Processors Interconnected for Concurrent Execution for Accelerating the SPICE Circuit Simulator Using an FPGA. IEEE Transactions On Computer-aided Design of Integrated Circuits and Systems 2012; 31(1) 9-22.

[16] Zorzi M, Franzè F, Speciale N. Construction of VHDL-AMS simulator in MATLABTM. In: Proceedings Of The 2003 International Workshop On Behavioral Modeling, 2003, San Jose, USA, 2002.

[17] Camera K. SF2VHD: A Stateflow to VHDL Translator. Master degrees. University of California, Berkeley, USA; 2001.

[18] Sbarcea B, Nicula D. Automatic Conversion of MATLAB/Simulink Models to HDL Models. http://www.fcd.co.il/doc/optim2004.pdf (accessed 20 September 2012).

[19] Mirotznik M S. Translating MATLAB programs into C code. IEEE Spectrum 1996; 33(1) 63-64.

[20] Shen S et al. Inferring Assertion for Complementary Synthesis. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 2012; 31(8) 1288-1292.

[21] Sinha R, Patel H D. synASM: A High-Level Synthesis Framework With Support for Parallel and Timed Constructs. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 2012; 31(10) 1508-1521.

[22] Liu H et al. Automatic Decoder Synthesis: Methods and Case Studies. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 2001; 31(9) 1319-1331.

[23] Poikonen J H, Lehtonen E, Laiho M. On Synthesis of Boolean Expressions for Memristive Devices Using Sequential Implication Logic. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 2012; 31(7) 1129-1134.

[24] Wu X et al. Research and Application of Code Automatic Generation Algorithm Based on Structured Flowchar. Journal of Software Engineering and Applications 2011; 4(9) 534-545.

[25] Lien W et al. Counter-Based Output Selection for Test Response Compaction. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 2013; 32(1) 152-164.

[26] Marculescu D, Li P. Guest editorial special section on PAR-CAD: parallel CAD algorithms and CAD for parallel architecture / systems. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 2012; 31(1) 7-8.

[27] Meloni L G P, Lenzi K. G. Performance Measurement Simulations for Analog-to-Digital Converters. IEEE Transactions on Latin America 2012; 10(1) 1168-1174.

[28] Wang L, Kazmierski T J. VHDL-AMS based genetic optimization of fuzzy logic controllers. The International Journal for Computation and Mathematics in Electrical and Electronic Engineering 2007; 26(2) 447-460.

[29] Almeida, T. S., Silva, A. C. R., Rossi, S. R. SF2HDL: A Computational Tool of State Transition Diagram Translation. Journal of Mechanical Engineering and Automation., v.3, p.78-86, 2013.

[30] Karris S T. Introduction to Stateflow with applications. United States: Orchard Publications; 2007.

[31] Tancredo L O. TAB2VHDL: Um Ambiente de Síntese Lógica para Máquinas de Estados Finitos. Master degrees. Universidade Estadual Paulista, Ilha Solteira, Brazil; 2002.

[32] 1364-2005-IEEE Standard for Verilog Hardware Description Language. http://standards.ieee.org/findstds/standard/1364-2005.html (accessed 25 December 2013).

[33] Design Entry and Synthesis. http://www.altera.com/products/software/quartus-ii/subscription-edition/design-entry-synthesis/qts-des-ent-syn.html (accessed 25 December 2013).

[34] Almeida, T. S., Silva, A. C. R., Grout, I. A. Acquired Experiences With Computational Tool MS2SV Used in Electronic Circuit Design. Electrical and Electronic Engineering., v.3, p.97-104, 2013.

[35] SystemVision Multi-Discipline Development Environment. http://www.mentor.com/products/sm/system_integration_simulati on_analysis/systemvision/ (accessed 25 December 2013).

[36] Silva A C R, Grout I A, Jeffrey R, O'Shea T, Generating VHDL-AMS Models of Digital-to-Analogue Converters From MATLAB/Simulink. In: Thermal, Mechanical and Multi-Physics Simulation Experiments in Microelectronic and Microsystems, 2007-EUROSIM 2007, 2007, London; 2007.

[37] Analog Devices Inc. CMOS dual 8-bit buffered multiplying DAC: AD7528. http://www. datasheetcatalog.org/datasheet/analogdevices/78586868AD 7528_b.pdf (accessed 5 September 2013).

[38] Tyagi A K. MATLAB and Simulink for Engineers. USA: Oxford University Press; 2012.

[39] Analog Devices Inc. CMOS 8-Bit Buffered Multiplying DAC: AD7524. http://www.datasheet catalog.org/datasheet/analogdevices/888358036ad7524.pdf (accessed 5 September 2013).

[40] Analog Devices Inc. 8-Bit, High-Speed, Multiplying D/A Converter (Universal Digital Logic Interface) DAC: DAC08. http://www.datasheetcata log.org/datasheet/analogdevices/80487346DAC08_b.pdf (accessed 5 September 2013).

[41] Oppenheim A V, Willsky A S, Hamid S. Signals and Systems. USA: Prentice Hall; 1996.

[42] Almeida, Tiago Da Silva, Silva, Alexandre Cesar R. Da, Sampaio, Daniel J. B. S. Computational Tool to Support Design of DAC Converter Model AD7528 with the Object Code in XML In: 2012 IEEE Electronics, Robotics and Automotive Mechanics Conference (CERMA), Cuernavaca. 2012 IEEE Ninth Electronics, Robotics and Automotive Mechanics Conference. IEEE, 2012. v.9. p.327 – 333.

[43] Cruz, E. L., Almeida, T. S., Silva, A. C. R. Metodologia para síntese automática de máquinas de estados finitos baseada em descrição em alto nível de abstração In: VIII International Conference on Engineering and Computer Education, 2013, Luanda. Forming Engineers for a Growing Demand. Santos: Claudio da Rocha Brito, 2013. v. 8. p.371-375