VHDL code generation based on a hierarchical node structure

ALIN BADEA, EMIL POP, FLORIN BADEA
University of Petrosani
Str. Universitatii nr. 20, Petrosani, HD
ROMANIA
alin.badea@omnisource.com, emilpop2001@yahoo.com, flo_badea@yahoo.com

Abstract – This paper presents a VHDL code generation algorithm based on a hierarchical structure of elements provided by a graphical design tool that allows the design and simulation of VLSI dedicated microcontrollers. The control algorithm is designed visually using a software application based on the concept of logic neural networks [1]. Once the algorithm is implemented using the design tool, structured VHDL code [4] is generated using the same application based on the algorithm described in this paper. The generated code can then be downloaded into a dedicated VLSI device.

Key-Words: VHDL, VLSI, MVVM, logic neuron, code generation, design tool.

1 Introduction

1.1 Hierarchical structure

The hierarchical structure used to generate the VHDL code is based on the concept of node [7]. A node is an element with multiple inputs and outputs and it can implement a variety of logic functions. In this particular case it implements the logic function of a logic neuron [7].

$$y = \bar{u} * x_0 + u * x_1 \qquad (1)$$

A node is represented as the concrete implementation of the INode interface. This interface is presented in fig.1a.

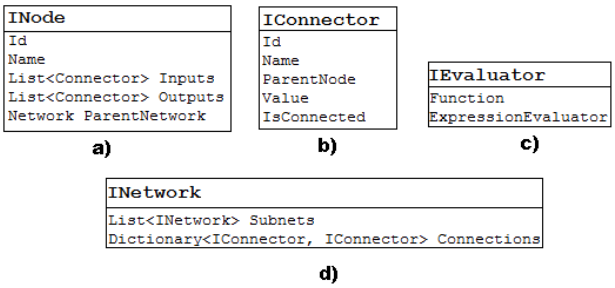

Fig.1 a) INode interface b) IConnector interface c) IEvaluator interface d) INetwork interface

The INode interface uses the IConnector interface (fig1.b) to represent the node inputs and outputs. An input or output can be seen as the concrete implementation of the IConnector interface.

The implementation of INode is based on another interface named IEvaluator (fig.1.c). This interface allows the implementation of the logic function specific of each node and provides an algorithm to evaluate this function to produce the node output.

The VHDL code generator will read the Function property to determine the architecture of each individual node in the care of simple nodes. In the case of nodes that represent complex structures like a whole network the generation algorithm will be recursively called for each node contained in the network.

The nodes along with their connections to each other will be stored in a network which is an implementation of the INetwork interface (fig.1.d). As can be seen from the previous figure a network can also contain one or more networks and the connections between them. This implementation detail will be used by the code generator to generate the signals that connect the design components inside a network.

1.2 VHDL code generation based on the hierarchical structure

The class that encapsulates the code generation algorithm is called VHDLCodeGenerator and has the following structure.

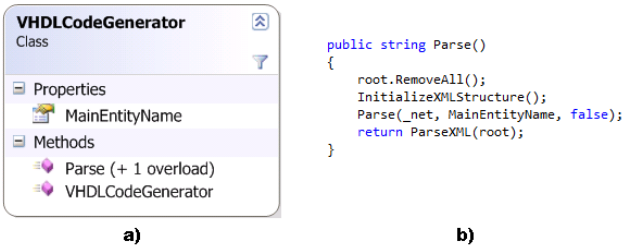

Fig.2 a) Structure of the VHDLCodeGenerator class b) Parse method implementation

This class exposes 2 public members, the MainEntityName property and the Parse method. The property allows getting and setting the name of the main entity. The Parse method implements the code generation algorithm.

The code generation algorithm has 2 steps. In the first step the structure of the network is read and an intermediary XML file is generated. In the second step the XML file is read and the VHDL code is generated. This code can then be compiled using the various tools available and downloaded into the dedicated VLSI device.

From the implementation of the Parse method (fig.2.b) we can see that the method is based on 2 private methods to run the 2 steps of the algorithm(Parse and ParseXML).

In the code above, root represents the root of the XML structure generated by the algorithm. This XML structure is reset each time the algorithm is run. The InitializeXMLStructure method initializes this structure as follows:

- It sets the name of the main entity;
- It sets up the inner structure of the main entity and its architecture, generating XML elements for the entity and its ports, the architecture, components, signals, constants, signal mappings and outputs.

The private Parse method generates an intermediary XML structure using the following algorithm.

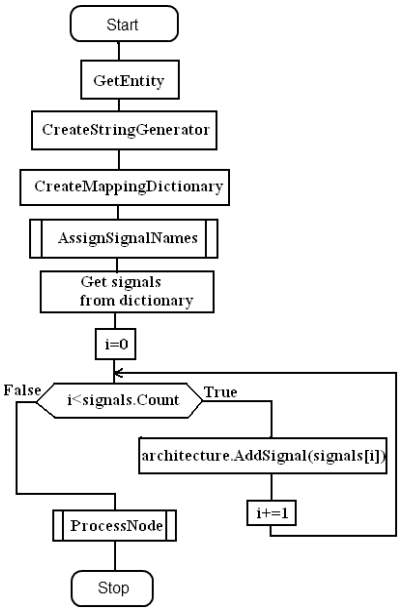

Fig.3 Logic diagram of the ProcessNetwork method

The AssignSignalNames and ProcessNode methods implement the following algorithms.

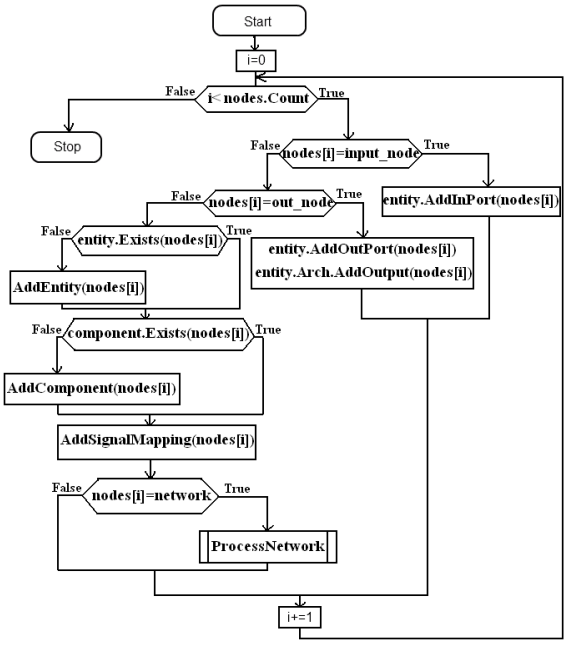

Fig.4 Logic diagram AssignSignalNames method

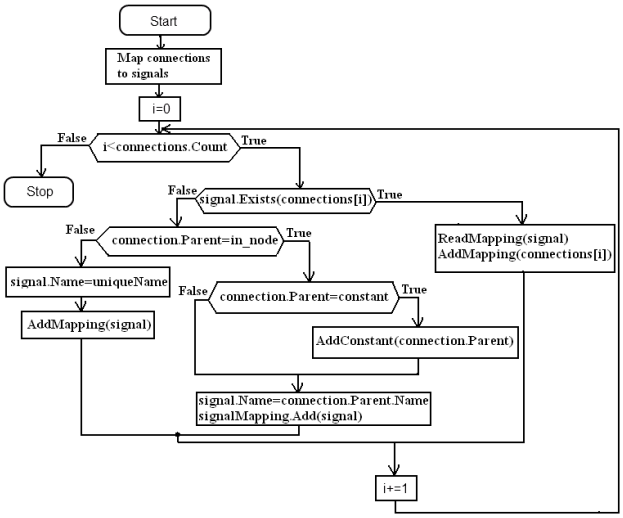

Fig.5 Logic diagram ProcessNode method

VHDL code generation based on the XML structure is done using the following algorithm.

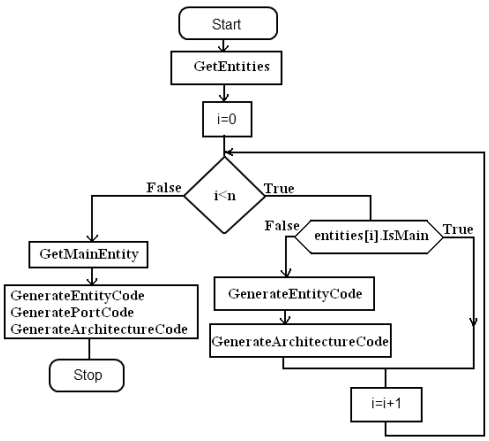

Fig.6 Logic diagram of the generation algorithm

2 The design application

2.1 Principles of design

This section describes a design tool used to design and simulate various algorithms that are to be programmed in VLSI dedicated microcontrollers. The application was implemented using the principles of OOP design and was architected using the MVVM pattern[5][6]. The design tool allows algorithms to be designed based on the concept of logic neurons. The GUI for the application is presented in the following figure.

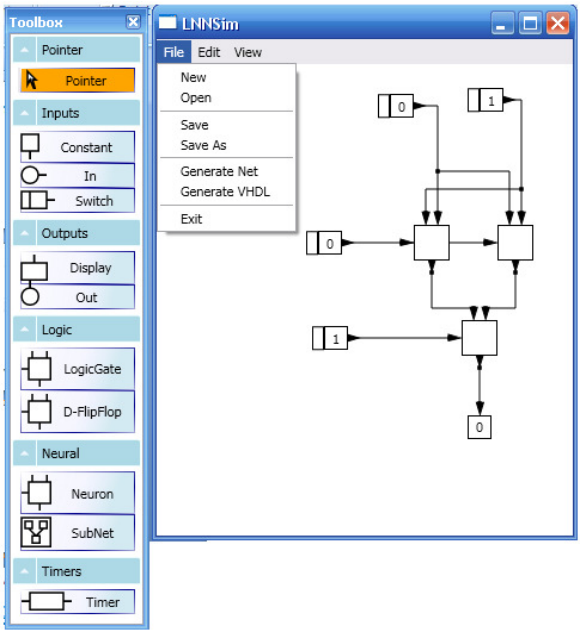

Fig.7 Designer tool main window

The design tool allows the user to place the various elements available in the toolbox onto a design surface and then to connect these elements together in order to create a more complex design.

The elements in the toolbox are based on the logic neuron element which is also present in the toolbox and can be used as it is. By connecting several elements together one can create a logic neural network that can then be simulated right in the designer.

After the logic neural network has been created, the application can be used to generate structured VHDL code based on this network, code that can be used to program any VLSI device.

2.2 Practical application

In order to demonstrate the code generation algorithm a control and monitoring algorithm for the extraction mining machine was designed and implemented using the design tool described previously. This application will then be used to generate the VHDL code based on the designed algorithm.

The logic diagram of the algorithm is presented in the following figure. Then the algorithm was implemented using the designed application.

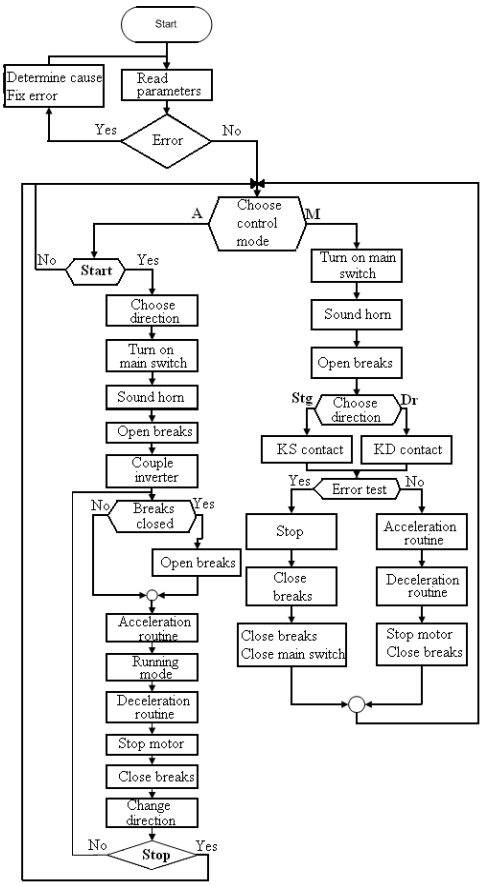

Fig.8 Algorithm logic diagram

The control algorithm's block diagram and the implementation of the algorithm using the designed application are presented in the following figures.

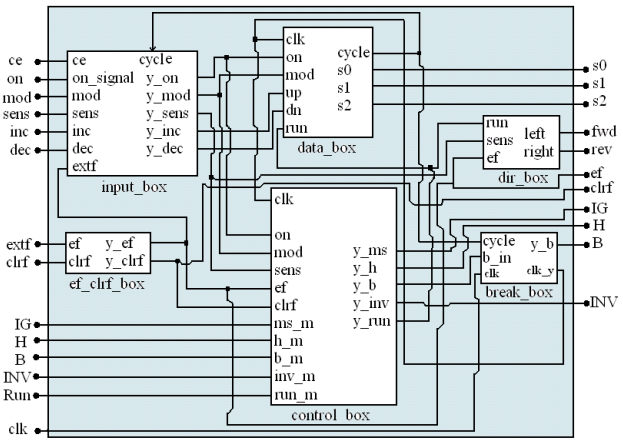

Fig.9 Algorithm block-diagram

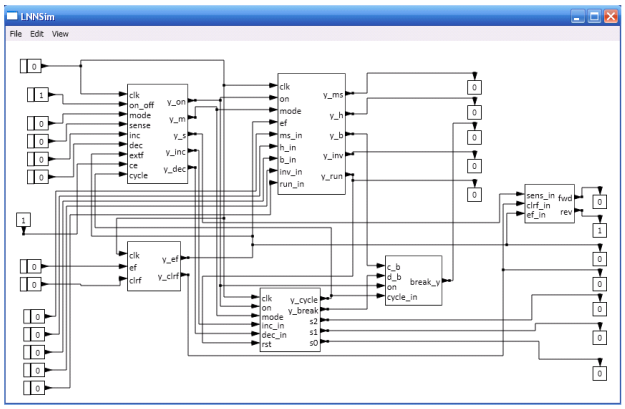

Fig.10 Algorithm implementation

The above block diagram describes an algorithm that allows both manual and automatic control of the extraction mining machine. In the following paragraphs a short description of the containing blocks is made.

The input_box block determines, based on the input signals, if the circuit is active, the control mode and the direction. When manual control is chosen the outputs y_inc and y_dec will be used to send the speed of the motor to the data_box. This block is active only if the "ce" signal has a value of logic 1. The implementation of this block is presented next.

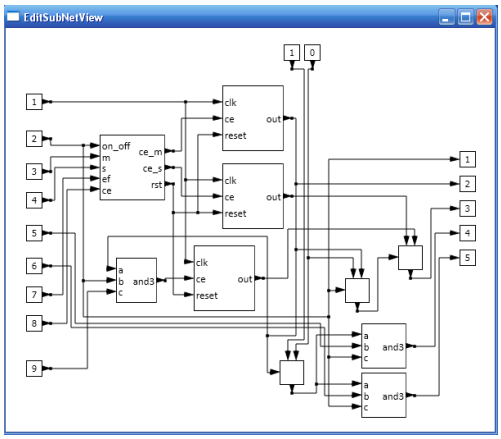

Fig.11 input_box block implementation box

The dir_box block determines the direction of the motor. This block is implemented as follows.

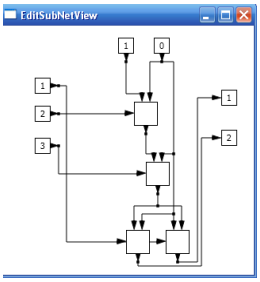

Fig.12 dir_box block implementation

The ef_clrf_box block determines the values of the External fault and Clear external fault signals based on the input signals. This block is implemented as follows.

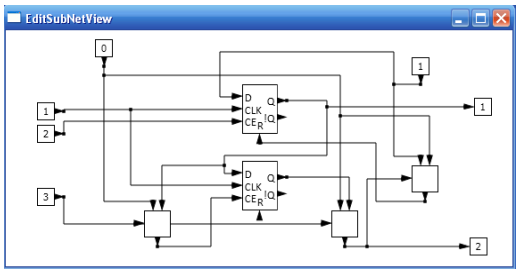

Fig.13 ef_clrf_box block implementation

The data_box block implements the control tahogram. If manual control is chosen then the up and dn inputs will be used to send the motor speed to the inverter (0-7 in binary coding). If automatic control is chosen the cycle output will determine the direction and the opening and closing of the breaks each time the tahogram terminates.

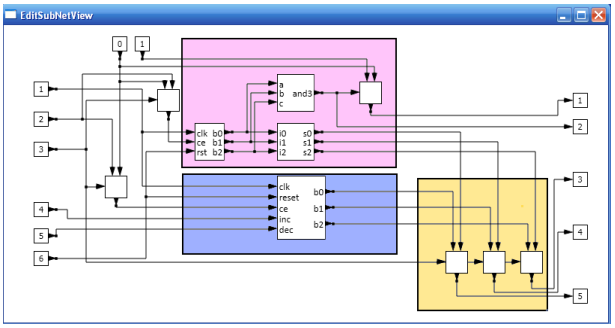

Fig.14 data_box implementation

As can be seen from the above figure this block is made up of 3 sections. The upper section is used when automatic control is chosen and allows the speed of the motor to be based on the imposed tahogram. This section is made up of an n-bit counter (left) and a block that implements the desired tahogram(right).

The lower section is used when manual control is chosen and allows setting the speed of the motor manually using the inc and dec inputs. This section is composed of a 3-bit reversible counter.

The section in the lower right part is used to determine which of the two blocks described above will be used to set the speed of the motor.

The control block is implemented as follows.

ISBN: 978-1-61804-118-0

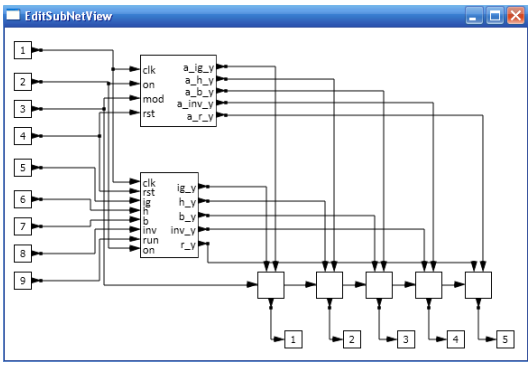

Fig.15 control_box block implementation

This block is made up of two sub-blocks as follows. One sub-block for when manual control is chosen (bottom part) and one block for when automatic control is chosen (top part). These are implemented as follows.

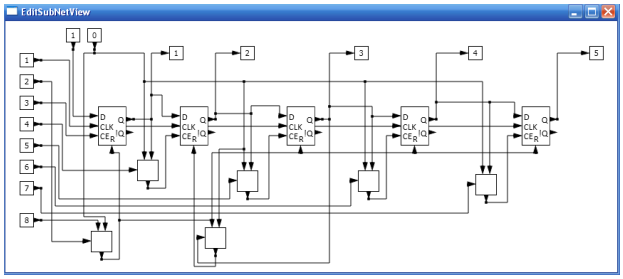

Fig.16 Manual control sub-block

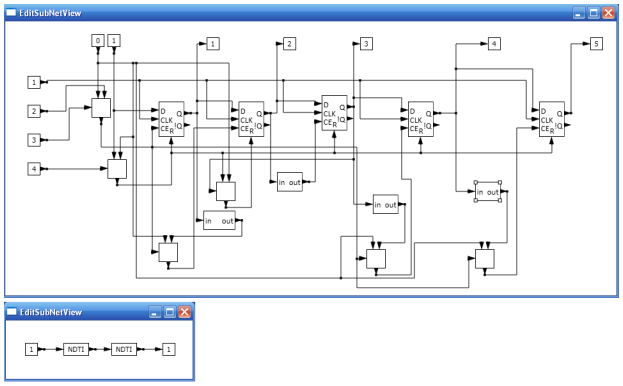

Fig.17 Automatic control sub-block

3 Simulation and results

Based on this algorithm the structured VHDL code was generated. The code was compiled and the following simulation scenarios were devised.

a) Manual control mode, direction left then right, the user can specify the speed of the motor using the inc and dec inputs.

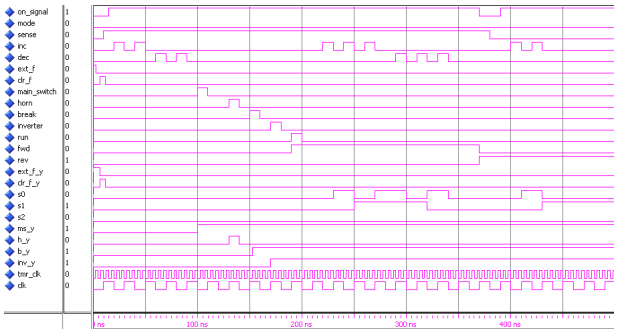

Fig.18 Simulation scenario no.1

b) Automatic control mode, direction right. In this case the direction changes automatically when the tahogram terminates.

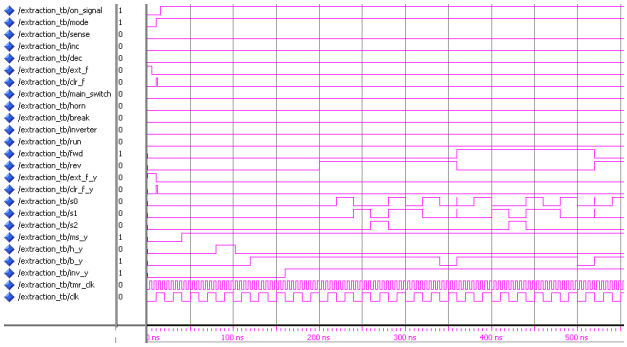

Fig.19 Simulation scenario no.2

c) Automatic control mode, then the simulation of an external fault, fixing the fault and restarting the machine.

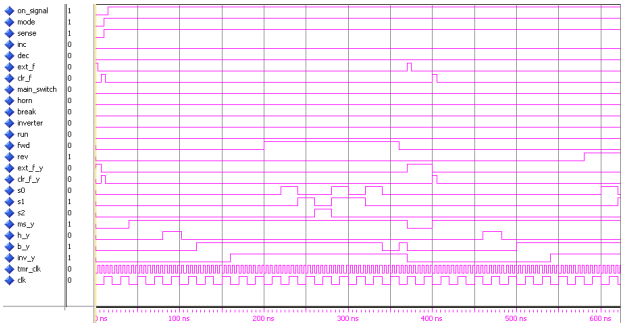

Fig.20 Simulation scenario no.3

After simulation the algorithm was programmed in a FPGA development kit.

4 Conclusions

This paper presents a VHDL code generation algorithm that takes a hierarchical structure of nodes representing a logic neural network as input and outputs structured VHDL code that can be compiled and downloaded into a VLSI device.

The paper also presents a short description of the application that is used to create the hierarchical

ISBN: 978-1-61804-118-0

node structure. The application allows the user to create a network by placing toolbox elements on the design surface and connecting them. The application also simulates the network and allows the generation of VHDL code based on it.
An algorithm was implemented that allows manual and automatic control of the extraction mining machine. If the algorithm is in the automatic control mode and an external fault occurs the algorithm will stop and default to the manual control mode in order to allow the human operator the possibility to reset the system.

References:

[1]. E. Pop, M. Leba, Microcontrollere şi automate programabile. Editura Didactică şi Pedagogică, Bucureşti, 2003.

[2]. M. MacDonald, Pro WPF in C# 2010, Apress, 2010.

[3]. B. Cohen, VHDL Coding Styles and Methodologies 2nd ed. Kluwer Academic Publishers, 2002.

[4]. Pong P. Chu, FPGA Prototyping by VHDL examples. Wiley Interscience, 2008, chapters 9, 12, 13.

[5]. Gossman J. Introduction to Model/View/ViewModel pattern for building WPF apps.

[6]. Gossman J. Advantages and disadvantages of M-V-VM

[7]. Pop E., Leba M., Pop M., Sochirca B., Badea A. – Software Based on Logic Neural Networks for Digital Controllers Design, Proceedings of the 8th WSEAS International Conference on CIRCUITS, SYSTEMS, ELECTRONICS, CONTROL AND SIGNAL PROCESSING (CSECS '09), Puerto de la Cruz, Tenerife, Canary Islands, Spain, ISBN 978-960-474-139-7, ISSN 1790-5117, pp. 168 -173, 2009