MODEL BASED HARDWARE DESIGN WITH SIMULINK HDL CODER

Krasimira Filipova [1], Tsvetomir Dimov [2]
[1] Technical University of Sofia, Faculty of Automation, 8 Kliment Ohridski, 1000 Sofia, Bulgaria, Phone: +359 2 965 25 26, E-mail: Krfil [AT] tu-sofia.bg
[2] Technical University of Sofia, Faculty of Automation, 8 Kliment Ohridski, 1000 Sofia, Bulgaria, E-mail: Tz.Dimov [AT] gmail.com

Abstract*:* The paper examines a method for model based design of hardware systems. Simulink models can be used for hardware design by using Simulink HDL Coder for HDL code generation. This method for electronic system development and automatic code generation have become an established technology in the design process. In this paper is presented an approach which will be shown with an example of traffic light model.

Key words: Simulink, HDL Code, FPGA

1. Introduction

Programmable devices (FPGA – Field Programmable Gate Array) are very important part of the development process for almost every electronic system. FPGA gives resources that can be configured to implement variety of arithmetic and logical functions. This resources include specialized DSP blocks, multipliers, memory, data structures, registers, tristate buffers, multipexors [1].

Model based design with Simulink gives an opportunity for obtaining hardware descriptions without handwriting of HDL code and by using an automatic code generation process. This can be done by Simulink HDL Coder and a special library of blocks that supports code generation.

Nowadays, model based development is common practice with a wide range of specialized software tools for modeling and simulation such as MATLAB and Simulink are used for specifying, designing, implementing, and checking the functionality of new controller functions. The quality and efficiency of the software are strongly dependent upon the quality of the model used for code generation.
The usage of Simulink can reduce the time for system design. This environment also gives error free code and no debugging is needed.

There are many architectures and implementation processes made by different software packages. In this article is used Xilinx ISE Project Navigator. This software package gives a convinient way for simulation of different system descriptions and synthesis of electronic systems which are described with hardware language. (HDL - Hardware Description Language). The two most popular hardware description languages are VHDL and Verilog [2]. In this article is used a discription in Verilog language.

This paper will present one of the most modern and effective methods for digital systems design and its main benefits.

2. Description of the method for HDL code generation

The only alternative to general purpose DSPs is to realise the algorithm into an ASIC for hardware acceleration. Normally, the system prototype is implemented on FPGA because of its flexibility and the final implementation will be on ASIC because of the high speed and low cost for a serial production.

The needed hardware description will be automaticly generated by using Simulink functionality for hardware design by abstractions, which can be automaticly compilated into HDL code. Another important functionality is the generation of a test bench to be used for simulation of the hardware description [3]. Model based design by Simulink is very flexible because of the easy

changes in the model. This approach makes the electronic system design a lot faster with no need for attention to internal connections in the device prototype. Simulink uses suitable libraries for eficent implementation of the functions, built with blocks. By using this design method it is not necessary to have detailed information for the characteristics of the used FPGA and easily can be made changes in the design [4]. Also important is the possibility for using the hole MATLAB functionality for data analisys and visualization in the design process.

The transition from Simulink model to hardware description is made by a group of blocks, united in a library (*hdllib/hdlsupported*). They are like the other Simulink blocks, but they represent specific components of a hardware device and from models built with blocks from this library can be generated HDL code. This automated transfer allows HDL language not to be learned and the code don't need debugging.

2.1. Simulink HDL Coder

Simulink HDL Coder automates the algorithm design process, from modeling to FPGA implementation. It generates bit-true and cycle-accurate, synthesizable Verilog and VHDL code from Simulink models, MATLAB code, and Stateflow charts [7]. The generated HDL code can be simulated and synthesized using standard tools and then implemented on FPGA. Simulink HDL Coder can control HDL architecture, implementation and generate hardware resource utilization reports. For rapid verification, Simulink HDL Coder generates test benches and EDA cosimulation models, and provides code traceability.

HDL code generation process starts by modeling the algorithm in Simulink, MATLAB, or Stateflow. HDL code can be generated from MATLAB code by using the Embedded MATLAB function block. Simulink HDL Coder provides a library of logic elements, such as counters and timers that are written in MATLAB code. Also finite-state machine can be modelled in Stateflow and handwritten or legacy HDL code can be integrated into the Simulink model via black-box interfaces.

Simulink HDL Coder generates VHDL and Verilog test benches to enable rapid verification of the generated HDL code. HDL testbench can be customized by a variety of options that apply stimuli to the HDL code. Also the process of compiling and simulating the code can be automated by generating a script.

The algorithms and designs used to define systems are normally modeled using high level software languages like MATLAB or C. But these designs could not be suited to real hardware implementation. Simulink HDL coder is a new tool, which comes with MATLAB-Simulink software package and can be used to generate HDL code based on Simulink models and Stateflow finite-state machines [5]. The coder brings the Model based design into the domain of applicationspecific integrated circuit (ASIC) and field programmable gate array (FPGA) development. Using the coder, system architects and designers can spend more time on tuning the algorithms and models and experimentation and less time on HDL code writing. Simulink HDL coder compatibility checker utility can be run to examine MATLAB-Simulink models and blocks for HDL code generation compatibility, then by invoking the coder, using either the command line or the graphical user interface.

The coder generates VHDL or Verilog code that implements the design embodied in the model. Usually, a test bench also can be generated. The test bench with HDL simulation tools can be used to simulate the generated HDL code and evaluate its behavior. The coder generates scripts that automate the process of compiling and simulating the. Various software packages can be used by the coder to cosimulate generated HDL entities within a Simulink model.

Simulink HDL Coder generates script files for use with HDL simulation and synthesis tools. Script generation is executed automatically when is enabled and code generation is started. By default, Simulink HDL Coder generates script files that are compatible with the Mentor Graphics ModelSim HDL simulator and with Synplicity Synplify synthesis software. Simulink HDL Coder can be programed to generate scripts for most EDA

tools. EDA script generation can be customized via the Simulink HDL Coder GUI, or by setting *makehdl* or *makehdltb* properties at the command line.

Due to the drastic reduction of design time, between the design developer and implementation engineer and the absence of design misunderstandings, this automatic method of converting an algorithm into a hardware design proved to be feasible.

2.2. Creation of a Simulink model

For automatic generation of hardware description and FPGA implementation the model have to be realised with blocks from the library *hdlsupported*. This library starts by the command *hdllib* in Matlab command line.

The traffic light model is built by blocks from this library. The model is shown on a Fig. 1.

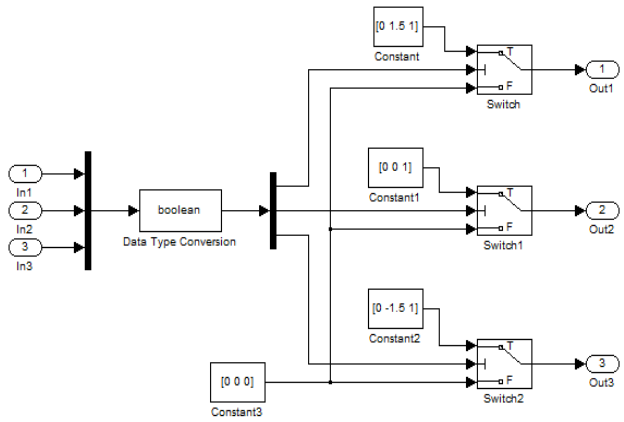

Fig. 1 Traffic light Simulink model

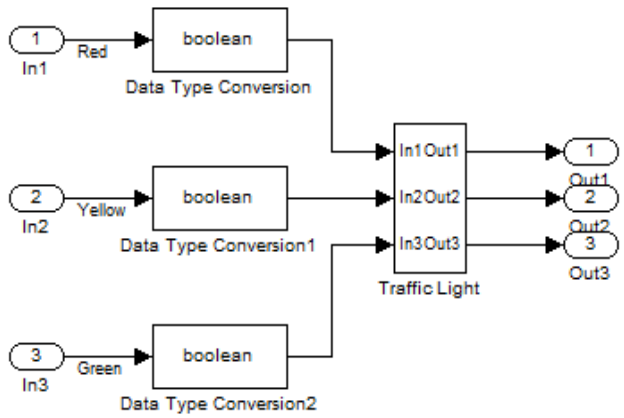

Fig. 2 Model with a subsystem

This is a relatevly small model. In a bigger projects the different parts of a model can be grouped in subsystems. By doing this a hardware description can be generated for parts of the system. The traffic light model can be presented with a subsystem. This is shown on a Fig. 2.

By double clicking on the sybsystem block (Traffic Light) it contents can be viewed as is shown on Fig. 1. Hardware descriptions of a different parts of a system or subsystems can be collected in the simulation stage. This is possible because of the hardware language standard. Also a hand written code parts can be added to a project.

2.3. HDL code generation

The HDL code generation starts from the Simulink menu Tools->HDL Code Generation->Generate HDL. The generated Verilog code is shown on Table 1.

Table 1. The generated HDL code.

Line	Verilog code
1	// ----------
2	--------
3	//
4	// File Name:
5	C:\Users\tzveto\Documents\tl\tlhdl.v
6	// Created: 2012-04-16 12:23:29
7	//
8	// Generated by MATLAB 7.13 and Simulink
9	HDL Coder 2.2
10	//
11	//
12	// -- ----------
13	----------
14	// -- Rate and Clocking Details
15	// -- ----------
16	----------
17	// Model base rate: 0.1
18	// Target subsystem base rate: 0.1
19	//
20	// ----------
21	--------
22	// ----------
23	--------
24	//
25	// Module: tlhdl
26	// Source Path: tlhdl
27	// Hierarchy Level: 0
28	//
29	// ----------

30	`timescale 1 ns / 1 ns

```
31  module tlhdl
32          (
33           In1,
34           In2,
35           In3,
36           Out1_0,
37           Out1_1,
38           Out1_2,
39           Out2_0,
40           Out2_1,
41           Out2_2,
42           Out3_0,
43           Out3_1,
44           Out3_2
45          );
46    input   In1;
47    input   In2;
48    input   In3;
49    output  [63:0] Out1_0;  // double
50    output  [63:0] Out1_1;  // double
51    output  [63:0] Out1_2;  // double
52    output  [63:0] Out2_0;  // double
53    output  [63:0] Out2_1;  // double
54    output  [63:0] Out2_2;  // double
55    output  [63:0] Out3_0;  // double
56    output  [63:0] Out3_1;  // double
57    output  [63:0] Out3_2;  // double
58
59    wire [63:0] Traffic_Light_out1_0;  // ufix64
60    wire [63:0] Traffic_Light_out1_1;  // ufix64
61    wire [63:0] Traffic_Light_out1_2;  // ufix64
62    wire [63:0] Traffic_Light_out2_0;  // ufix64
63    wire [63:0] Traffic_Light_out2_1;  // ufix64
64    wire [63:0] Traffic_Light_out2_2;  // ufix64
65    wire [63:0] Traffic_Light_out3_0;  // ufix64
66    wire [63:0] Traffic_Light_out3_1;  // ufix64
67    wire [63:0] Traffic_Light_out3_2;  // ufix64
68    // <Root>/Traffic Light
69    //
70    // <Root>/Data Type Conversion
71    //
72    // <Root>/Data Type Conversion1
73    //
74    // <Root>/Data Type Conversion2
75    Traffic_Light   u_Traffic_Light   (.In1(In1),
76                                       .In2(In2),
77                                       .In3(In3),
78
79    .Out1_0(Traffic_Light_out1_0),     //  double
80    .Out1_1(Traffic_Light_out1_1),     //  double
81    .Out1_2(Traffic_Light_out1_2),     //  double
82    .Out2_0(Traffic_Light_out2_0),     //  double
83    .Out2_1(Traffic_Light_out2_1),     //  double
84    .Out2_2(Traffic_Light_out2_2),     //  double
85    .Out3_0(Traffic_Light_out3_0),     //  double
86    .Out3_1(Traffic_Light_out3_1),     //  double
87    .Out3_2(Traffic_Light_out3_2)      //  double
88    );
89      assign Out1_0 = Traffic_Light_out1_0;
90      assign Out1_1 = Traffic_Light_out1_1;
91      assign Out1_2 = Traffic_Light_out1_2;
92      assign Out2_0 = Traffic_Light_out2_0;
93      assign Out2_1 = Traffic_Light_out2_1;
94      assign Out2_2 = Traffic_Light_out2_2;
95      assign Out3_0 = Traffic_Light_out3_0;
96      assign Out3_1 = Traffic_Light_out3_1;
97      assign Out3_2 = Traffic_Light_out3_2;
98    endmodule  // tlhdl
```

When the generated code is written it is ready for simulation. Also additional files for various software simulators can be generated for the next step when the code will be simulated.

2.4. Simulation of the generated code

After the hardware description is generated the simulation can be performed by using the Xilinx software ISE Project Navigator. The HDL code (Verilog) is added to the project in this software package. The code is checked for errors, simulated and the RTL scheme realisation can be viewed by selecting View RTL Schematic.

After syntax check (Behavioral Syntax Check) the simulation of the behavioral model is performed by selecting Simulate Behavioral Model [6].

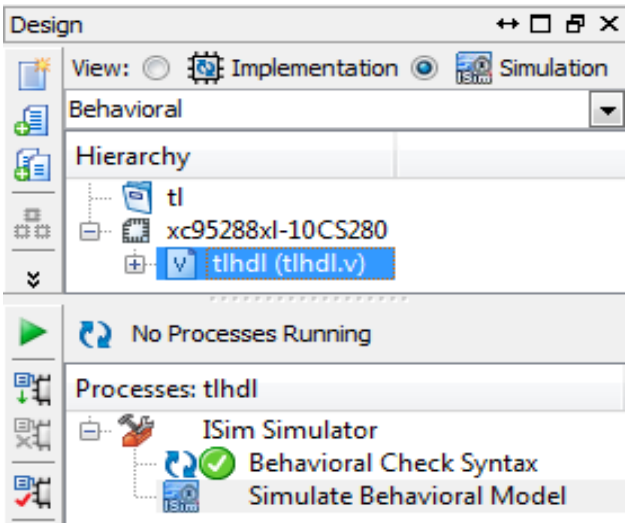

Fig. 3 Simulation in Xilinx ISE

The figure above (Fig. 3) shows the interface of Xilinx ISE software. After behavioral syntax

check the model can be simulated. If there are no errors the simulation results are shown.

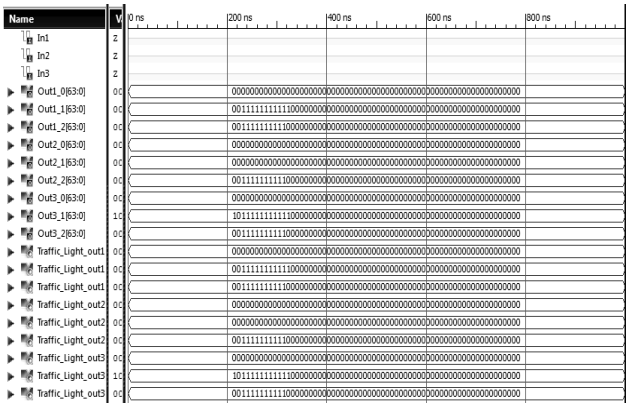

Fig. 4 Simulation results

From the simulation results (Fig.4) of the hardware description can be made the conclusion that the generated hardware description is accurate.

2.5. Results analysis.

The generated HDL code from a Simulink model gives values that are acurate in time and can be simulated and synthezed with variety of software tools and after that can be implemented in FPGA [7]. Simulink also can control the HDL architecture and implementation, and to generate a report for the used resources. For a vast verification it can generate a test files and cosimulation models and gives code traceability. This design method from modeling to programmable device implementation becomes recursive.

By this method the behavior and synthesis of a various types of systems can be simulated [8]. It also gives to a system designers an easy way to add inputs to the designed system, whitch are called test vectors and to observe the new output reactions. With this, by simulation, the designer can assure of the accurate performing of the designed system by passing the input data and observing the output.

3. Conclusion

The modern programmable devices in combination with appropriate software packages for synthesis and simulation give a significantly accelerated design process of electronic systems.

This approach for automatic generation of a hardware descriptions block built models of functions and systems saves significant amount of time for code writing, debugging and verification of the generated hardware.

The conclusion that can be made is this approach is suitable for hardware generation by model based design in Simulink. It also gives potentiality for simultaneous design, simulation, analysis and visualisation by Matlab and Simulink [9]. All this gives contribution to the development of the hardware based system design.

Acknowledgement

The results, published in this paper, are gotten by the project № 112pd055-8 financed by the Technical University of Sofia – Fund Scientific Investigations.

References:

[1] I. Grout, *Digital systems design with FPGAs*. Elsevier Ltd., 2008, Oxford.

[2] К. Филипова, М. Христов, *Използване на (v)HDL за синтез на електронен хардуер*, Издателство КИНГ-2001, 2004, София.

[3] C. Maxfield, *The design warrior's guide to FPGAs*, Elsevier Ltd., 2004, Oxford.

[4] B. Zeidman, *Introduction to CPLD and FPGA Design*, The Chalkboard Network, 2001.

[5] Kr. Filipova, Ts. Dimov, D. Djamijkova, F. F. Filipov, "Showing the capabilities of VHDL description and Simulink HDL Coder for control system", IX INTERNATIONAL CONFERENCE "CHALENGES in HIGHER EDUCATION and RESEARCH in 21st CENTURY", Sozopol, Bulgaria, 2011.

[6] Kr. Filipova, Vl. Yankov, F. F. Filipov, Y. Kralev, Ts. Dimov, "Investigating opportunities for hardware realization of transfer functions", Sixth International Conference - Computer Science'2011, Ohrid, R. Macedonia, 2011.

[7] I. Petrinska, F. Filipov, Ts. Dimov, K. Filipova, "Intelligent Lighting Control System for Education Buildings", Lux junior, 10. Forum für den lichttechnischen Nachwuchs, Dörnfeld bei Ilmenau, Germany, 2011.

[8] Xilinx, "ISE Design Suite" http://www.xilinx.com/support/documentation/index.htm

[9] MathWorks, "Simulink HDL Coder", http://www.mathworks.com/products/slhdlcoder/