Showing the capabilities of VHDL description and Simulink® HDL Coder for control system

Krasimira Filipova [1], Tsvetomir Dimov [2], Filip Filipov [3], Denitca Djamijkova [4]

[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
[3] Technical University - Sofia, FDIBA, Bulgaria, E-mail: pilif.pilif [AT] gmail.com
[4] Technical University – Vienna, Austria, E-mail: Denitsa.Djamiykova [AT] gmail.com

Abstract: This paper shows the capabilities of VHDL description and the usage of Simulink HDL Coder for description of electronic control system without studying the electric circuits in detail. This is demonstrated by two examples, based on VHDL code description and Simulink HDL Coder. The results are achieved by block diagram of algorithm for behavior description of the system or by a logical function as a base for a model of digital electronic circuit.

Key words: Hardware Description Languages, VHDL, Simulation, Simulink, Conversion

1. Introduction

This paper describes a suitable way to convert every digital algorithm, in corresponding electronic hardware. Several Hardware Description Languages (HDLs) are utilized within the design process for these digital systems. Formalized approaches may provide a solution whose effectiveness can be analyzed and where improvements to both the design approach and chosen implementation architectures can be predicted.[1] [2]

With the use of a VHDL or Simulink HDL Coder the FPGA Design can be automated created without circuit implementation as shown on the figures below. [3] [4] [5]

Important is the possibility to control the post synthesis timing report and to annotate back the Simulink model to identify timing-constraint bottlenecks. Such integration with synthesis tools provide rapid design iterations and significantly reduce FPGA design cycle time. The overall intention is to model the behavior of a digital algorithm with given user-defined parameters and to write directly the VHDL code or to create automatically in Simulink the code and to convert this automatically to VHDL code after that.[6] [7] [8] [9].

2. Praxis with VHDL Code

The usage of VHDL, MATLAB/Simulink HDL coder and the new toolboxes for description of the algorithm is examinated. Also, it's possible for this approach to use the capabilities of mentioned toolboxes for automated generation of HDL code, independently of writing the VHDL code. The approach, used in this paper, at first, is to use manually creation of the VHDL code for the control unit to be designed. [11] [12] [13]

At the beginning we have to describe special features and how it works. Below, on fig.1 is shown the block diagram of the investigated algorithm. It contains 8 logical blocks, 9 inputs and 7 outputs. On the shown algorithm we have to virtually synthesize the electric control unit. For this we have to follow two basic steps:

creation of the VHDL code

creation of the testbench for testing the sequence of events, for correctness of this sequence and for estimation of the algorithm correctness.

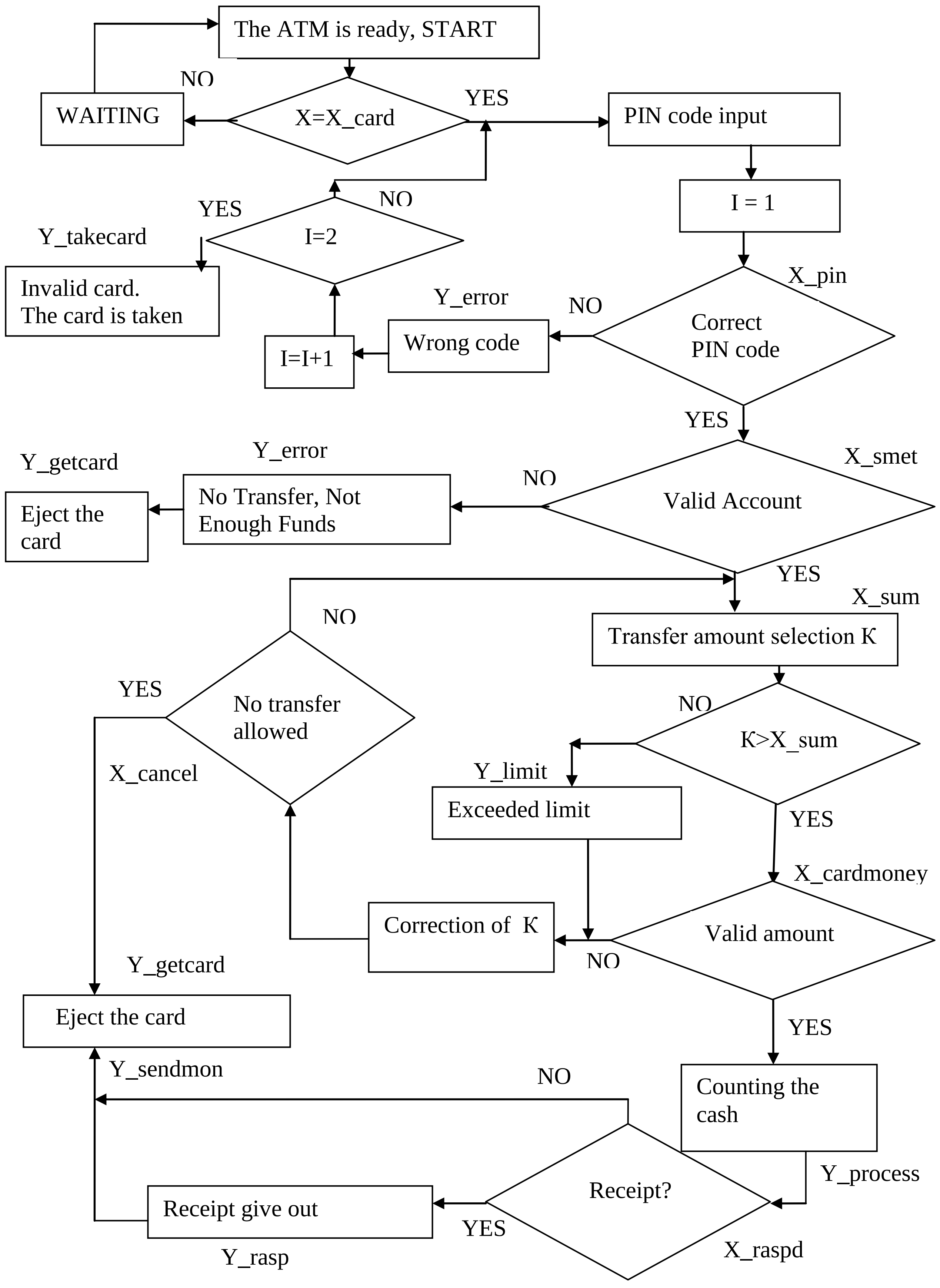
The ATM is ready, START
NO
WAITING
X=X_card
YES
PIN code input
I = 1
YES
NO
I=2
Y_takecard
Invalid card. The card is taken
X_pin
Y_error
NO
Correct PIN code
Wrong code
I=I+1
YES
Y_getcard
Y_error
X_smet
NO
No Transfer, Not Enough Funds
Eject the card
Valid Account
YES
X_sum
NO
Transfer amount selection К
YES
No transfer allowed
NO
К>X_sum
X_cancel
Y_limit
Exceeded limit
YES
X_cardmoney
Correction of К
Valid amount
NO
Y_getcard
Eject the card
YES
Y_sendmon
NO
Counting the cash
Y_process
Receipt?
Receipt give out
YES
X_raspd
Y_rasp

Fig. 1

On Fig. 2 is shown the VHDL code. It consists of 98 rows.

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity logic is
 port(
     clk, reset: in std_logic;
          x_card, x_pin, x_smet,x_cancel, x_raspd,
x_cardmoney: in std_logic;
          x_sum: in unsigned(8 downto 0);
          y_error, y_takecard, y_limit,y_process,
y_sendmon,y_rasp, y_getcard: out std_logic
          );
end logic;

architecture Behavioral of logic is

signal k : unsigned(8 downto 0);

type statetype is (idle,start, sec, tree, break);
   signal state_reg: statetype;
   signal pin: std_logic;

begin

process(clk,reset)
variable i: integer :=1;
begin
     if reset='1' then
       state_reg <= idle;
            i:=1;
            k <= "110010000";
            pin <='1';
          elsif rising_edge(clk)  then
          case state_reg is
        when idle =>
               y_error <='0';
               y_takecard <='0';
                    y_process <='0';
                    y_limit <= '0';
                    y_getcard<='0';
                    y_rasp<='0';
                    y_sendmon <='0';
               if (x_card='1') then
               state_reg <= start;
               end if;
                when start =>
          if (x_pin = pin) then
                 state_reg <= sec;
                 y_error <='0';
                 elsif (i <3) then
                    i:=i+1;
                    y_error <='1';
                    else
                    y_takecard <='1';
                    state_reg <= idle;
                    end if;
     when sec =>
          if (x_smet='1') then
          if (k > x_sum or k= x_sum ) then
          if (x_cardmoney= '1') then
          y_process <='1';
          state_reg <=tree;
          end if;
          else
          y_limit <= '1';
          if (x_cancel = '1') then
          state_reg <= break;
          end if;
          end if;
          else y_error<='1';
          y_getcard<='1';
          state_reg <=idle;
          end if;
      when tree =>
                    if (x_raspd = '1') then
                    y_rasp<='1';
                    state_reg <= break;
                              y_sendmon <='1';
                    else
                         state_reg <= break;
                                   y_sendmon <='1';
                    end if;

      when break =>

      y_getcard <='1';
      state_reg <= idle;
      end case;
      end if;
      end process;

end Behavioral;
```

Fig. 2.

On Fig. 3 is shown the testbench. It's obvious that the algorithm is realized, when the corresponding output signals occur, indicating for their assumptive random events.

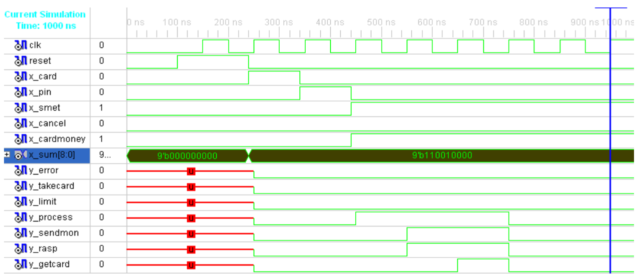

Fig. 3

In the testbench the physical delays haven't a reflex. In this paper we assume these delays haven't an impact on the execution of the sequence of actions. This special case is examined by the authors in another paper.

The creation and verification of the code for control of the standard automated transfer machine shows, that including more conditions and making its work more complicated is possible and on principle is the same. In this way we can say that the virtually created electronics for control are standard and unified. The control system is implemented on a programmable devices as CPLD and FPGA . By doing this a big part of the electronics became integrated in the device and this significantly eases the design of the control system.

3. Praxis with automated generation of HDL code

The other possibility for code generation by automated synthesis with implementation on programmable devices is shown too.

On Fig. 4 is shown a logical scheme that realizes the Boolean function:

$$Y = (\overline{X_0} \vee X_2 \vee X_3)(\overline{X_0} \vee \overline{X_1} \vee X_3)$$
$$(X_0 \vee X_1 \vee \overline{X_2} \vee X_3)(\overline{X_0} \vee X_1 \vee \overline{X_2} \vee X_3)$$

Electronic circuit like this can be examined as a realization of the block-diagram of another algorithm. For this scheme as a specific distinction from the algorithm on Fig. 1, that in this case the operations are with signals. By this simple scheme is demonstrated that always is possible to expand the block-scheme of the control algorithm and make it more complicated.

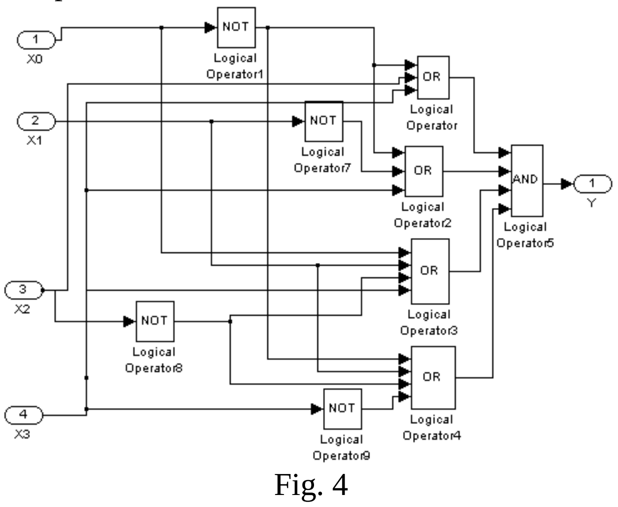

Fig. 4

On Fig. 5 is shown the VHDL code automatically generated via MATLAB/Simulink HDL coder, and on Fig.6 is shown the testbench for estimation of the code correctness. This paper doesn't make a comparison between the effectiveness of the synthesis by manually writing the VHDL code (it's necessary to be aware of code semantics and typing rules) and the automatic code generation via MATLAB/Simulink HDL coder (in this case it's necessary to know this big software product and to have some experience). Both ways are followed by testbench verification as the resource consumption is almost the same.

Of course, the processing and synthesis of control by block-scheme algorithms are forthcoming and not only for discrete-event systems.

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY opit1_2_1 IS
 PORT( X0                    :   IN    std_logic;
       X1                    :   IN    std_logic;
       X2                    :   IN    std_logic;
       X3                    :   IN    std_logic;
       Y                     :   OUT   std_logic
       );
END opit1_2_1;

ARCHITECTURE rtl OF opit1_2_1 IS
 -- Signals
 SIGNAL X0_1                       : std_logic;
 SIGNAL Logical_Operator1_out1          : std_logic;
 SIGNAL X1_1                       : std_logic;
 SIGNAL Logical_Operator1_out1_1        : std_logic;
 SIGNAL X2_1                       : std_logic;
 SIGNAL X0_2                       : std_logic;
 SIGNAL X3_1                       : std_logic;
 SIGNAL Logical_Operator1_out1_2        : std_logic;
 SIGNAL Logical_Operator_out1           : std_logic;

BEGIN
 X0_1 <=  NOT X0;
 Logical_Operator1_out1 <= X3 OR (X0_1 OR X2);
 X1_1 <=  NOT X1;
 Logical_Operator1_out1_1 <= X3 OR (X0_1 OR
X1_1);
 X2_1 <=  NOT X2;
 X0_2 <= X3 OR (X2_1 OR (X0 OR X1));
 X3_1 <=  NOT X3;
 Logical_Operator1_out1_2 <= X3_1 OR (X2_1 OR
(X0_1 OR X1));
 Logical_Operator_out1 <= Logical_Operator1_out1_2
AND (X0_2 AND (Logical_Operator1_out1 AND
Logical_Operator1_out1_1));
 Y <= Logical_Operator_out1;
END rtl;
```

Fig. 5

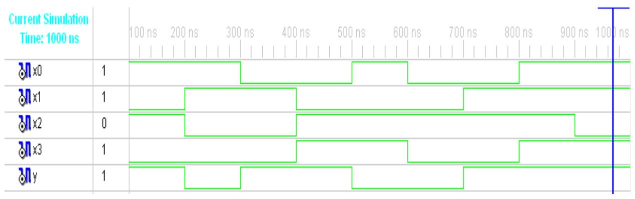

Fig.6

4. Conclusions and future work

The results show that for the synthesis of electronic control for different applications the needed code can be written down or generated. In both ways with a testbench the logical base shown in the block diagrams of the algorithm can be verified. These results give the opportunity for using this approach in the educational and research practice.

The research results presented in this paper are funded by Technical University – Sofia research project No. 112pd055-8.

REFERENCES

[1] Chu, Pong P., "RTL Hardware Design using VHDL", John Wiley, 2006, ISBN 978-0-471-72092-8

[2] Kuon, Ian et al., "FPGA architecture", Now Publishers Inc, 2008, ISBN 978-1601981264

[3] Samilagic, Zoran S., "Digital systems design and prototyping using field programmable logic", Springer, 2000, ISBN 978-0792379201

[4] Roth, Charles H. et al, "Digital systems design using VHDL", CL-Engineering, 2007, ISBN 978-0534384623

[5] Brown, Stephen et al. "Fundamentals of Digital Logic with VHDL Design", McGraw-Hill, 2008, ISBN 978-0077221430

[6] Hwang, E., "Digital Logic and Microprocessor Design with VHDL", CL-Engineering, 2005, ISBN 978-0534465933

[7] Lee, S., "Advanced Digital Logic Design Using VHDL, State Machines, and Synthesis for FPGA's", CL-Engineering, 2005, ISBN 978-0534466022

[8] Chartrand, L., "Digital Fundamentals: Experiments and Concepts with CPLDs", 2003, ISBN 978-1401842468

[9] Dueck, R., "Digital Design with CPLD Applications and VHDL", 2004, ISBN 978-1401840303

[10] Filipova, Kr., Petrakieva, S., Filipov, F., Costov, I., "Hardware Realization of the Control Algorithm in Hydrosystems with FPGA", Proc. International Conference "Computer Science'08", Kawala , pp115-120

[11] Mladenov, V., Filipova, Kr., Petrakieva, S., Dimov, B., Uhlmann, F., "Analysis of Signal Competition in Asynchronous Ultra High-Speed Digital Circuits", Przeglad Elektrotechniczny (Electrical Review), Issue 11, 2007, ISSN 0033-2097, Poland, pp. 197-200

[12] Stoyadinova, T., Buzov, Il., Mladenov, V., Filipova, Kr., Ortlepp, T., Panayotov, I., "Development of VHDL-models for transient simulation of complex asynchronous RSFQ circuits" Paper ID: 209 , IWK 51 , Ilmenau, 2009