VLSI Embedded Solution for Multi-Drive Conveyors Control

Adrian Avram, Emil Pop, and Camelia Barbu

***Abstract*—** In this paper is presented the controller modeling, simulation and implementation the VLSI approach. In order to apply this solution there is analyzed the case of multi-drive conveyor controller. This controller is designed using the VHDL language. The VLSI version is good for high complexity systems and for prototyping.

Keywords— controller, conveyor, multi-drive, VHDL, VLSI.

I. Introduction

MANY times, for technical reasons, the conveyors are driven using multiple motors. In order to start such a driving system must be used a sequential algorithm having a certain time step between each motor starting. This algorithm makes the controller complex because each motor's run cycle must be controlled in order to reach the steady state speed, besides the sequential control. Also, it is necessary to monitor several technological conditions, like: conveyor discharge, conveyor regime speed and emergency state.

In figure 1 there is presented such a conveyor driven by three motors (M_1, M_2, M_3) monitored by three transducers for: discharge (TD), speed (TS) and emergency (TE).

Before staring any motor there must be used a klaxon (KLX) few seconds signaling.

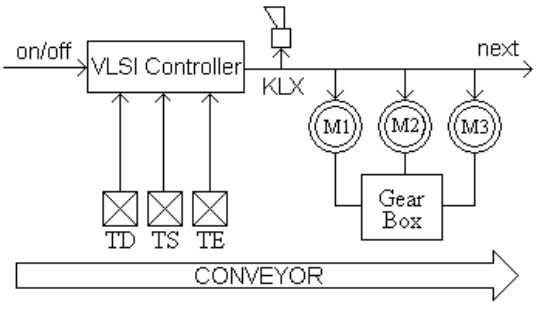


Fig.1 Conveyor control system block diagram

The control algorithm must satisfy the following conditions:

- Using the on/off buttons the system can be manually stopped.
- If, during the system running, a motor stops, then all the system is stopped.
- After the start signal on/off there must be performed the system integrity test, by polling the transducers.
- There is signaled for 5...10 seconds the warning alert by KLX.
- There are started sequentially by a preset time step the three motors M_i.
- There is tested the conveyor's speed using a watchdog timer (WD). If the speed is too low, then all the system is stopped.
- There are tested the transducers TD and TE that must be off, meaning that there is no damage. If any of these transducers are on, then all the system is stopped.

Manuscript received August 19, 2010.

A.Avram is Ph.D. student at the University of Petrosani, Romania.

E.Pop is with the University of Petrosani, Romania (tel: +40 743172943, e-mail: emilpop@upet.ro, emilpop2001@yahoo.com).

C.Barbu is with the University of Petrosani, Romania (e-mail: tabacarucamelia@yahoo.com).

II. Problem Formulation

We will present at the beginning the bloc diagram of the control algorithm and after that we will present the control algorithm modeling and simulation.

A. The Control Algorithm

The algorithm block diagram is presented in figure 2.

First, we will model the algorithm in MatLab-Simulink environment. Based on the simulation results, we will implement the control algorithm in two ways: using a PLC and as embedded VLSI solution.

B. Modeling and Simulation

In this section, the mathematical model of the multi-drive conveyors control system will be presented. The model is then implemented in MatLab-Simulink and the simulation results are presented.

First, there are defined the notations used in the mathematical model: on/off – start/stop signals; t_e, t_d, t_s – signals from the emergency, discharge and speed transducer; wd – watch dog signal; T_d, T_w, T_{klx} – timings for drive start delay, watchdog and klaxon; y_1 – signal to start klaxon; y_2, y_3, y_4 – signals to start the three motor drives, y_5 – signal to start the next process, if any; $y_{on/off}$ – signal from the on/off buttons; y_{stop} – signal to stop the process; y_{no} – signal from the normal open contact of the relay; y_{noto} – signal from the normal open timing on opening contact of the relay; $\theta(t)$ – step distribution.

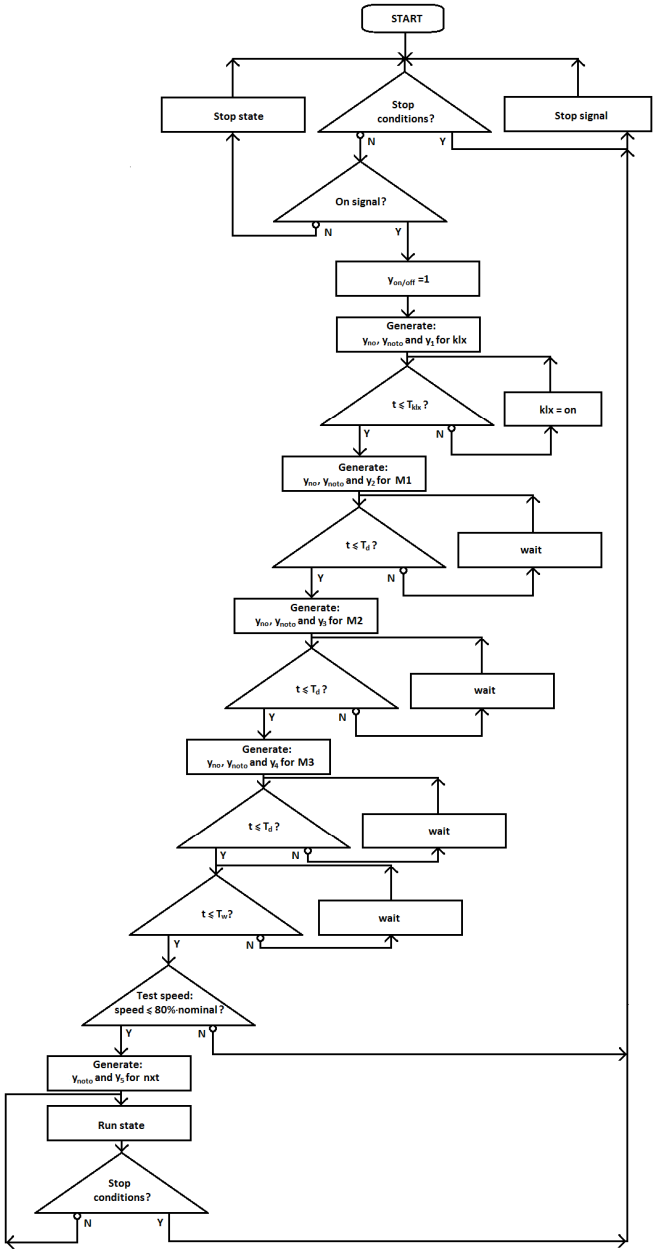


Fig.2 Algorithm block diagram

The logic equations representing the multi-drive system control are the following:

$$y_{on/off} = \overline{off} \cdot (on + y_{no}) \cdot y_{stop} \quad (1)$$

$$y_{stop} = \left(\overline{y_2 \cdot y_3 \cdot y_4 \cdot y_5} + t_s + wd\right) \cdot \overline{t_e} \cdot \overline{t_d} \quad (2)$$

$$y_{noto} = u \cdot [\theta(t) - \theta(t - T_d)],\ u \in \{y_{on/off}, y_2, y_3, y_4, y_5\} \quad (3)$$

$$wd = y_{noto} \cdot [\theta(t) - \theta(t - T_w)] \quad (4)$$

$$y_1 = y_{noto} \cdot y_{on/off} \quad (5)$$

$$y_2 = y_{no} \cdot \overline{y_{noto}} \cdot y_{on/off} \quad (6)$$

$$y_3 = y_{no} \cdot \overline{y_{noto}} \cdot y_2 \quad (7)$$

$$y_4 = y_{no} \cdot \overline{y_{noto}} \cdot y_3 \quad (8)$$

$$y_5 = y_{noto} \cdot y_4 \quad (9)$$

Based on the algorithm and on the logic equations there was developed a MatLab-Simulink model as presented in fig.3.

In this figure, the on/off button and t_s, t_d and t_e were simulated by manual switches. The logic operations were implemented by the relations:

$$x\,and\,y = x \cdot y \quad (10)$$

$$x\,or\,y = x + y - x \cdot y \quad (11)$$

$$not\,x = \overline{x} = 1 - x \quad (12)$$

The time functions are represented by NOTO (normal open timing on opening) block for each necessary delay. The simulation results show in fig.4 the signals generated on start and stop conditions.

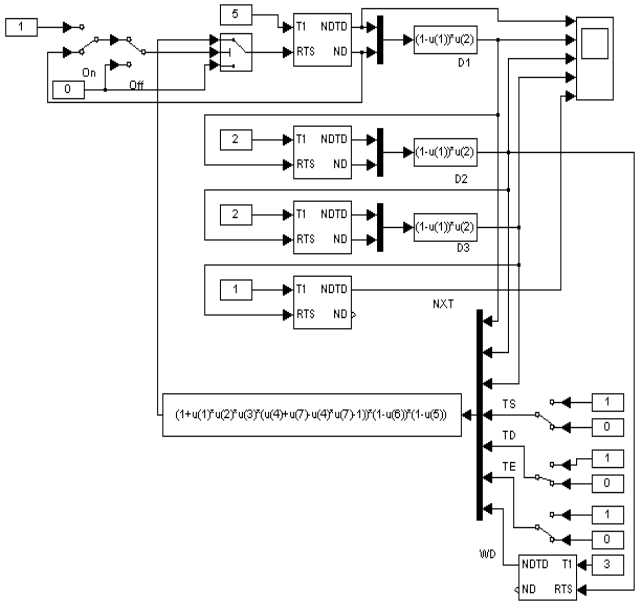


Fig.3 MatLab model

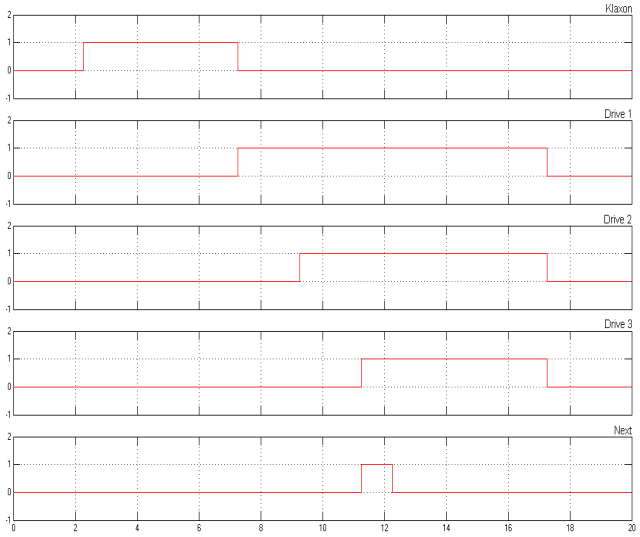


Fig.4 Simulation results

III. EMBEDDED VLSI SOLUTION

The VLSI implementation uses the VHDL language. The VHDL code was downloaded in a CPLD device. The block diagram of the CPLD based VLSI controller is presented in fig.5. The VHDL code is shown in fig.6.a and the simulation results in fig.6.b.

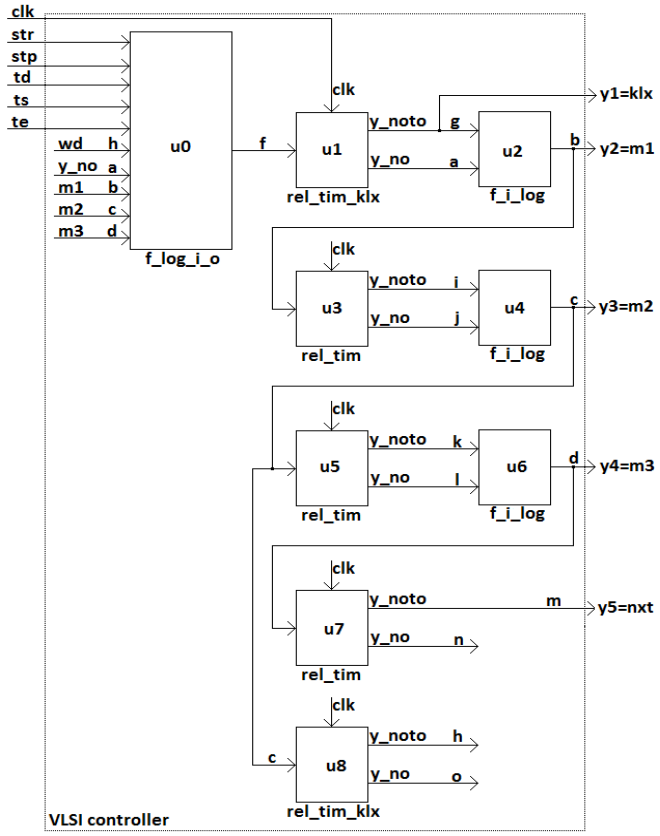


Fig.5 VLSI controller block diagram

As can be seen from the simulation results, the VLSI implementation works with very high accuracy and it is identically with the real process.

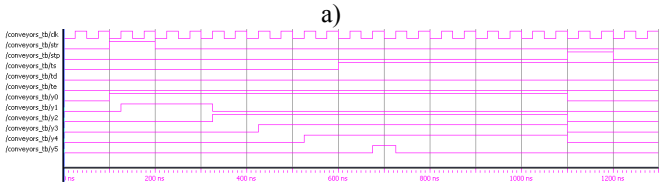
```
-----comp f_i_log_i_o------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity f_log_i_o is
        Port ( y_no : in  STD_LOGIC;
               str : in  STD_LOGIC;
               stp : in  STD_LOGIC;
               t_s : in  STD_LOGIC;
               t_d : in  STD_LOGIC;
               t_e : in  STD_LOGIC;
               w_d : in  STD_LOGIC;
               m1 : in  STD_LOGIC;
               m2 : in  STD_LOGIC;
               m3 : in  STD_LOGIC;
               f_str_stp : out  STD_LOGIC);
        end f_log_i_o;
architecture f_log_i_o of f_log_i_o is
begin
f_str_stp<=(not(m1 and m2 and m3) or
t_s or w_d)and (not(t_d)) and (not(t_e))
and (not(stp))and (str or y_no);
end f_log_i_o;
```

```
--------comp f_i_log-------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity f_i_log is
        Port ( y_noto : in  STD_LOGIC;
               y_no : in  STD_LOGIC;
               y : out  STD_LOGIC);
        end f_i_log;
architecture  f_i_log of f_i_log is
begin
y<=not(y_noto)and y_no;
end f_i_log;
--------comp rel_tim_klx------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity rel_tim_klx is
    Port ( clk : in  STD_LOGIC;
           u : in  STD_LOGIC;
           y_noto : out  STD_LOGIC;
           y_no : out  STD_LOGIC);
end rel_tim_klx;
architecture rel_tim_klx of rel_tim_klx is
constant T:integer:=5;
type state is(s0,s1,s2);
signal sc:state;
---------n-i---------
begin
process(clk,u)
variable count:integer range 0 to 100;
        begin
        if (u='1') then
                sc<=s1;
                count:=0;
        elsif(clk'event and clk='1') then
                count:=count+1;
                if (count=T) then
                        sc<=s2;
                        count:=0;
                        sc<=s0;
                end if;
        end if;
end process;
 ------n-s-------
          process(u,sc)
                  begin
                          case sc is
                          when s0=>
                          y_noto<='0';
                          y_no<='0';
                          when s1=>
                          y_noto<='1';
                          y_no<='1';
                          if (u='0')then
                          y_noto<='1';
                          y_no<='0';
                          end if;
                          when s2=>
                          y_noto<='1';
                          y_no<='0';
                  end case;
          end process;
 end rel_tim_klx;
 -------comp rel_tim------
 library IEEE;
 use IEEE.STD_LOGIC_1164.ALL;
 use IEEE.STD_LOGIC_ARITH.ALL;
 use IEEE.STD_LOGIC_UNSIGNED.ALL;
 entity rel_tim is
     Port ( clk : in  STD_LOGIC;
            u : in  STD_LOGIC;
            y_noto : out  STD_LOGIC;
            y_no : out  STD_LOGIC);
 end rel_tim;
 architecture rel_tim of rel_tim is
 constant T:integer:=2;
 type state is(s0,s1,s2);
 signal sc:state;
--------n-i----------
begin
process(clk,u)
       variable count:integer range 0 to 100;
               begin
               if (u='1') then
                       sc<=s1;
                       count:=0;
               elsif(clk'event and clk='1') then
               count:=count+1;
               if (count=T) then
                       sc<=s2;
                       count:=0;
                       sc<=s0;
               end if;
       end if;
end process;
```

```
-------n-s---------
        process(u,sc)
                begin
                case sc is
                        when s0=>
                        y_noto<='0';
                        y_no<='0';
                        when s1=>
                        y_noto<='1';
                        y_no<='1';
                        if (u='0')then
                        y_noto<='1';
                        y_no<='0';
                        end if;
                        when s2=>
                        y_noto<='1';
                        y_no<='0';
                end case;
        end process;
end rel_tim;
------- main --------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity controller_3tr is
        Port    ( clk : in  STD_LOGIC;
                str : in  STD_LOGIC;
                stp : in  STD_LOGIC;
                ts : in  STD_LOGIC;
                td : in  STD_LOGIC;
                te : in  STD_LOGIC;
                y0 : in  STD_LOGIC;
                y1 : out  STD_LOGIC;
                y2 : out  STD_LOGIC;
                y3 : out  STD_LOGIC;
                y4 : out  STD_LOGIC;
                y5 : out  STD_LOGIC);
end controller_3tr;
architecture Structural of controller_3tr is
component f_log_i_o is
        Port    ( y_no : in  STD_LOGIC;
                str : in  STD_LOGIC;
                stp : in  STD_LOGIC;
                t_s : in  STD_LOGIC;
                t_d : in  STD_LOGIC;
                t_e : in  STD_LOGIC;
                w_d : in  STD_LOGIC;
                m1 : in  STD_LOGIC;
                m2 : in  STD_LOGIC;
                m3 : in  STD_LOGIC;
                f_str_stp : out  STD_LOGIC);
end component;
component f_i_log is
    Port ( y_noto : in  STD_LOGIC;
           y_no : in  STD_LOGIC;
           y : out  STD_LOGIC);
end component;
component rel_tim_klx is
    Port ( clk : in  STD_LOGIC;
           u : in  STD_LOGIC;
           y_noto : out  STD_LOGIC;
           y_no : out  STD_LOGIC);
end component;
```

```
component rel_tim is
    Port ( clk : in  STD_LOGIC;
           u : in  STD_LOGIC;
           y_noto : out  STD_LOGIC;
           y_no : out  STD_LOGIC);
end component;
signal a,b,c,d,f,g,h,i,j,k,l,m,n,o: STD_LOGIC;
begin
u0 : f_log_i_o port
map(a,str,stp,ts,td,te,h,b,c,d,f);
u1 :  rel_tim_klx port map(clk,f,g,a);
u2 : f_i_log port map(g,a,b);
u3 : rel_tim port map(clk,b,i,j);
u4 : f_i_log port map(i,j,c);
u5 : rel_tim port map(clk,c,k,l);
u6 : f_i_log port map(k,l,d);
u7 : rel_tim port map(clk,d,m,n);
u8 : rel_tim_klx port map(clk,c,h,o);
y0<=f;
y1<=g;
y2<=b;
y3<=c;
y4<=d;
y5<=m;
end Structural;
```

a)



b)

Fig.6 Controller implementation: a) VHDL code; b) simulation results

IV. Conclusion

The VLSI version is very good for the achievement of high complexity control algorithms both as logic equations number and as inputs/outputs number. Also, it is suitable for development systems and prototyping.

The above conclusions were proved for the multi-drive conveyor control algorithm, which has a small to medium complexity and a small number of input/output signals. But for a complex system, with a very large inputs/outputs number, the VLSI control algorithm works identically, with minimum changes.

References

[1] E.Pop. *Automatizări în industria minieră*. Editura Didactică şi Pedagogică, 1983

[2] Pop E., Neghina F., Leba M., Barbu C. *Controlul adaptiv al sistemelor automate*. Editura Didactică şi Pedagogică, 2010

[3] Pop E., Covaciu C., Avram A., Neghina F. (2009). "Adaptive Control Strategy for Conveyor Drive Systems" presented at the 8th WSEAS International Conference on Circuits, Systems, Electronics, Control & Signal Processing, Puerto de la Cruz, Spain, ISBN 978 960 474 139 7, ISSN 1790 5117.

[4] Leba M., Badea F., Ciodaru D., Padurariu E. "Software Control and Monitoring for Underground to Surface Transport Process", presented at the 8th WSEAS International Conference on CIRCUITS, SYSTEMS, ELECTRONICS, CONTROL & SIGNAL PROCESSING (CSECS '09), ISBN 978 960 474 139 7, ISSN 1790 5117, Puerto de la Cruz, Tenerife, Canary Islands, Spain, pp. 150–155, 2009

[5] Volnei Pedroni, *Circuit Design with VHDL*. MIT Press, Massachusetts, USA, 2004.