International Journal of Control and Automation
Vol.9, No.2 (2016), pp.373-382
http://dx.doi.org/10.14257/ijca.2016.9.2.35

HDL Implementation of a Compositional Microprogram Control Unit with e-DIA

Young-Jin Oh and Gi-Yong Song[1]

School of Electronics Engineering, Chungbuk National University, Cheongju, Chungbuk, 362-763, Korea
{goodmen913, gysong} @ chungbuk.ac.kr

Abstract

In order to reduce hardware amount in a control unit, the CMCU (compositional microprogram control unit) that makes use of a property of operational linear chain by adopting counter into next-state logic, and DIA (dedicated input area) that splits control memory into two parts having different microinstruction formats are reviewed, and then the concept of TMI field of a DIA is extended to e-DIA in order to supplement the operation of a DIA. The occurrence of idle cycles is removed and microoperation missing or duplication in some cases is protected by applying the e-DIA to CMCU. CMCU with e-DIA is implemented using VHDL, and the improvements are validated through simulation.

Keywords*: Compositional microprogram control unit, Control memory, dedicated input area, e-DIA, VHDL*

1. Introduction

Interpretation and implementation of control algorithms for digital systems are based on finite-state machines or microprogram control units. The finite-state machine implementing control algorithms for digital system is represented by sequential circuit consisting of combinational circuit CC and register RG, with microinstruction sequence appearing on the output of RG. Implementation of a FSM (finite-state machine) logic circuit includes several steps such as derivation of state graph, state assignment, construction of structure table, determination of next state and output function and configuration of logic circuit using some logical elements [1]. In microprogram control units, each instruction of a high-level programming language is interpreted into a special microprogram that is kept in a separate CM (control memory) as a sequence of microinstructions. The design of microprogram control unit includes the following steps such as generation of microinstructions with a certain format, microinstruction addressing, construction of control memory content and synthesis of logic circuit. Composition of the finite-state machine and microprogram unit leads to a CMCU (compositional microprogram control unit) that has several particularities[2]. Among them are microinstruction format with operational part only, minimum length microprogram and multidirectional transition in one cycle of operation. In CMCU, the input address is generated by the block CC, and some additional block for address generation needs to be adopted to reduce the number of outputs of block CC. The CMCU with DIA(dedicated input area) adopted additional control memory for that purpose. The introduction of DIA causes idle cycles in the datapath in some cases. In this paper, e-DIA is proposed that removes idle cycles as well as preventing some microoperations from missing.

[1] Corresponding Author : gysong@chungbuk.ac.kr

ISSN: 2005-4297 IJCA

2. Compositional Microprogram Control Unit

The properties of control algorithm to be executed in a control unit have great influence on the hardware amount of corresponding control unit. One of such properties is operational linear chain included in control algorithm. A sequence of only operator vertices is called an OLC (operational linear chain) [1]. CMCU - one of the approaches implementing OLCs effectively - is a composition of an FSM and microprogram control unit [2]. The structural diagram of a CMCU is shown in Figure 1.

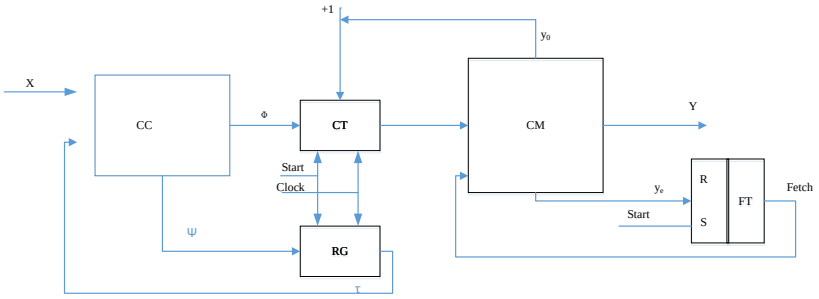

Figure 1. Structural Diagram of a CMCU

Combinational circuit CC and register RG form an FSM for microinstruction addressing. Counter CT, control memory CM and start-stop flip-flop FT form a microprogram control unit. Microinstructions for an OLC are placed in CM area with consecutive address, and are fetched out sequentially for producing output signals. On reaching the output vertex of an OLC, y_0=0 is issued and CC generates an initial address of an OLC that is to be executed next, and that address is loaded into CT. At the same time CC determines the code of next state to be loaded into RG. Let us show the procedure for CMCU synthesis with an FCA (flow-chart of an algorithm) 1 in Figure 2.

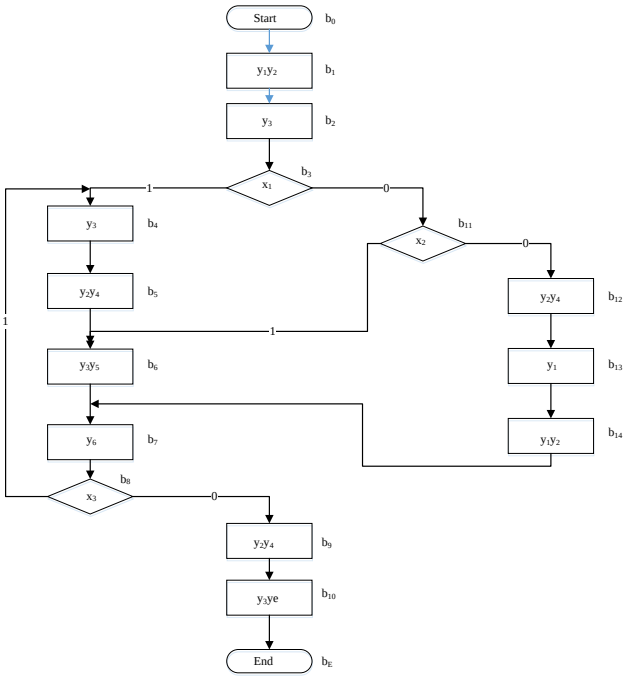

Figure 2. Flow-Chart of an Algorithm 1

The FCA1 includes four OLCs that are represented by a set c = {a_1… a_4}, where a_1= <b_1, b_2>, a_2 = <b_4, b_5, b_6, b_7>, a_3, = <b_{12}, b_{13}, b_{14}>, a_4 = <b_9, b_{10}>. First operator vertex of each OLC is assigned a specific address, and the other operator vertices in the same OLC are assigned addresses in a trivial way incrementing current address by 1. The contents of CM of CMCU implementing FCA1 is shown in Table 1.

Table 1. Control Memory Contents of CMCU

Address	Content
0000	y_0,y_1,y_2
0001	y_3
0010	y_0,y_3
0011	y_0,y_2,y_4
0100	y_0,y_3,y_5
0101	y_6
0110	y_0,y_2,y_4
0111	y_0,y_1
1000	y1,y_2
1001	y_0,y_2,y_4
1010	y_e,y_3

Address in case of transition between OLCs is specified by a state transition of a corresponding FSM. Interpretation of FCA1 as an FSM that specifies address in case of transition between OLCs leads to the transformed FCA1 shown in Figure 3 where I^i_j is i-th input of an OLC_j.

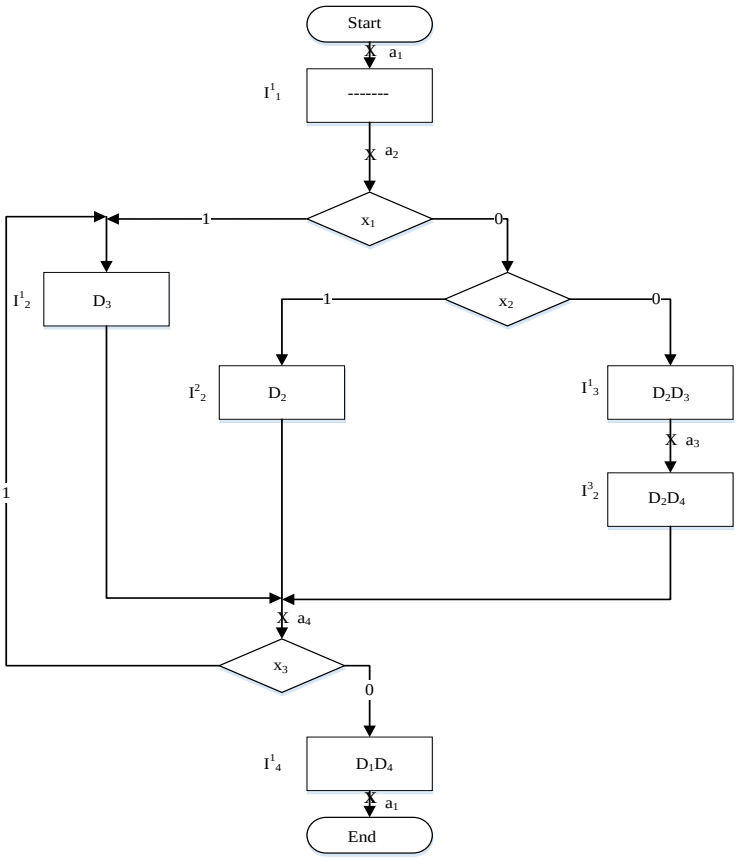

Figure 3. Transformed FCA 1

Transformed FCA1 representing a Mealy FSM includes four states a_i's where i=1, 2, 3 and 4. Each state is encoded from 00 in a trivial way. Information about state transitions such as next state and CM address for corresponding transition is included in the structure table of FSM of CMCU implementing FCA1 shown in Table 2.

Table 2. Structure Table of FSM of CMCU

a_m	$K(a_m)$	a_s	$K(a_s)$	X_h	ϕ_h	ψ_h
a_1	00	a_2	01	1	–	D_6
a_2	01	a_4	11	x_1	D_3	D_5D_6
		a_4	11	$\bar{x}_1x_2$	D_2	D_5D_6
		a_3	10	$\bar{x}_1\bar{x}_2$	D_2D_3	D_5
a_3	10	a_4	11	1	D_2D_4	D_5D_6
a_4	11	a_4	11	x_3	D_3	D_5D_6
		a_1	00	$\bar{x}_3$	D_1D_4	–

3. CMCU with E-DIA

In CMCU with DIA [2-4], DIA is a set of I_0 cells of CM having addresses from 0 to $(I_0\text{-}1)_2$, where I_0 is the number of OLC input in the FCA. The microinstruction format is shown in Figure 4, and the structural diagram of CMCU with DIA in Figure 5.

(a) Format for CM (b) Format for CM_0

Figure 4. Format of a Control Microinstruction for CMCU with DIA

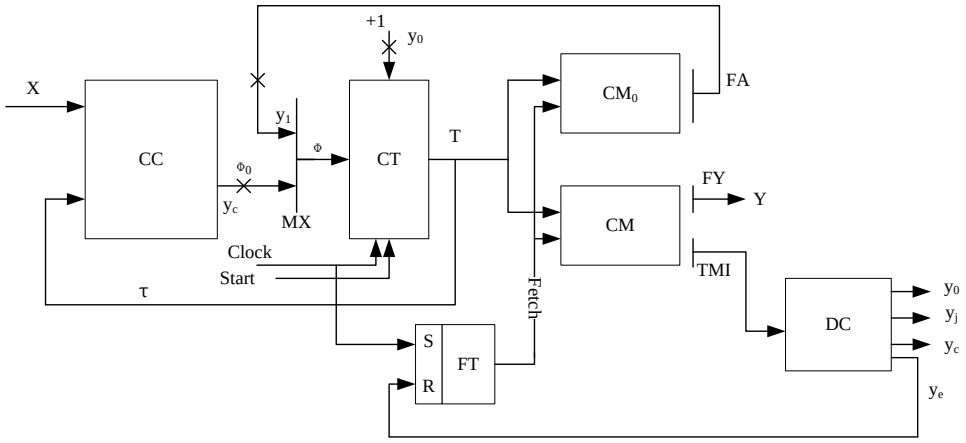

Figure 5. Structural Diagram of A CMCU with DIA

Compositional microprogram control unit in Figure 5 operates as follows. With the application of a start signal, input address of the fist microinstruction is loaded into the counter CT, and then first microinstruction is fetched from CM. The corresponding microoperations are issued, and at the same time the code of TMI is decoded into control signals y_0, y_j, y_c or y_e. Address sequence proceeds according to this control signal from

each microinstruction until y_e for termination of execution is issued with microoperations corresponding to each address being issued.

CM_0 keeping transition addresses is introduced in addition to CM in order to overlap microoperation issue and fetch of microinstruction address. The fundamental type of CMCU with DIA has CM only, and requires a cycle time to fetch a transition address without microoperation issue. The configuration of control memory with CM and CM_0 makes it possible to generate microoperations using the code from field FY at the same time fetching transition address using the code from field FA. FA is an address field with address for transition from DIA into AMP (area of microprogram) that keeps microinstruction for operator vertices following OLC input.

The field TMI determines the source of next microinstruction address. If TMI = '00', transition address is fetched from CM_0. If TMI = '01', next address becomes current address + 1. If TMI = '10', next address is determined from CC through function Φ_0, and points to DIA area. If TMI = '11', next address is set to 0 indicating termination of the execution of an FCA. TMI codes '00', '01', '10' and '11' are decoded into y_j, y_0, y_c or y_e in DC block of Figure 5, respectively.

In CMCU with DIA, field TMI from DIA area of CM is exclusively occupied with '00', indicating that next microinstruction address is always to be fetched from CM_0, and then loaded into CT without exception. This restriction on code selection of TMI field sometimes results in an idle cycle, or microoperation missing or duplication during execution of FCA. In e-DIA(extended - dedicated input area), field TMI is extended to codes other than '00', and the contents of corresponding location of AMP area is coordinated in accordance with the TMI code in e-DIA. As a result, both idle cycle and microoperation missing or duplication do not occur, leading to a smooth and correct microoperation issue. Let us demonstrate the operation of a CMCU with e-DIA with a simple example shown in Figure 6[2] to examine the role of e-DIA.

The e-DIA consists of OLC input vertices marked with a circle in Figure 6. These operator vertices are assigned address in such a trivial way that first vertex is assigned address 0, second vertex address 1, and so on. After OLC input vertices are assigned addresses, the other operator vertices are assigned addresses in the same manner starting from next of the end of e-DIA address range, building up an AMP area. Contents of CM includes both e-DIA and AMP, and is fixed as shown in Table 3 in the case of FCA2.

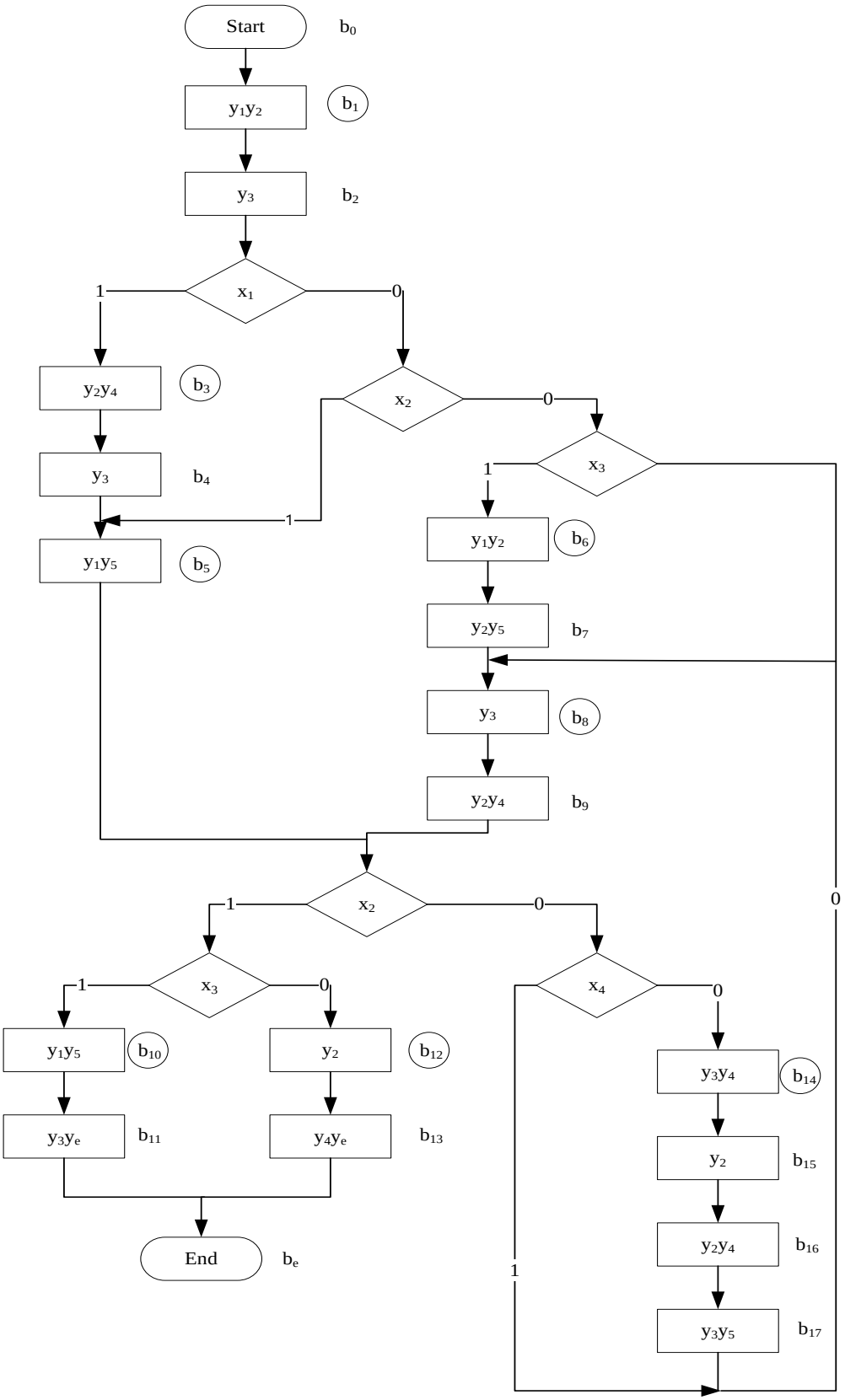

Figure 6. Flow-Chart of an Algorithm 2

Table 3. Contents of CM for CMCU with E-DIA

Address $T_1T_2T_3T_4T_5$	Content $m_1m_2m_3m_4m_5m_6m_7$
00000	0011000
00001	0001010
00010	1010001
00011	0011000
00100	0000100
00101	0010001
00110	0001000
00111	0000110
01000	1000100
01001	0100100
01010	1010001
01011	0101001
01100	0100100
01101	1001010
01110	1100100
01111	1100010
10000	0101000
10001	0101010
10010	1000101

First eight locations are occupied by e-DIA and the others by AMP. Each location of CM has TMI code and microoperation for processing of FCA2. Contents of CM_0 is fixed as shown in Table 4.

Table 4. Contents of CM_0 for CMCU with e-DIA

Address $T_3T_4T_5$	Content $a_1a_2a_3a_4a_5$
000	01000
001	01001
010	01010
011	01011
100	01101
101	01110
110	01111
111	10000

The column address represents the addresses of OLC inputs with minimal bits, and column contents keeps transition addresses from corresponding OLC input vertices to next operator vertices in CM.

In e-DIA, field TMI is not exclusively occupied by code ‘00’, and at the same time some microinstructions appear at location of e-DIA area as well as at location of corresponding AMP area to coordinate the generation of microoperations when execution of FCA pass through operator vertex that is an OLC input as well as an OLC output. A CMCU with e-DIA having CM and CM_0 corresponding to FCA2 was implemented using VHDL [5], and then simulations [6] are performed for several cases to check for correctness of microoperation issue. Simulation waveviews for some cases are shown in Figure 7 and Figure 8. The source code is shown in Appendix, and also can be referred to at [7].

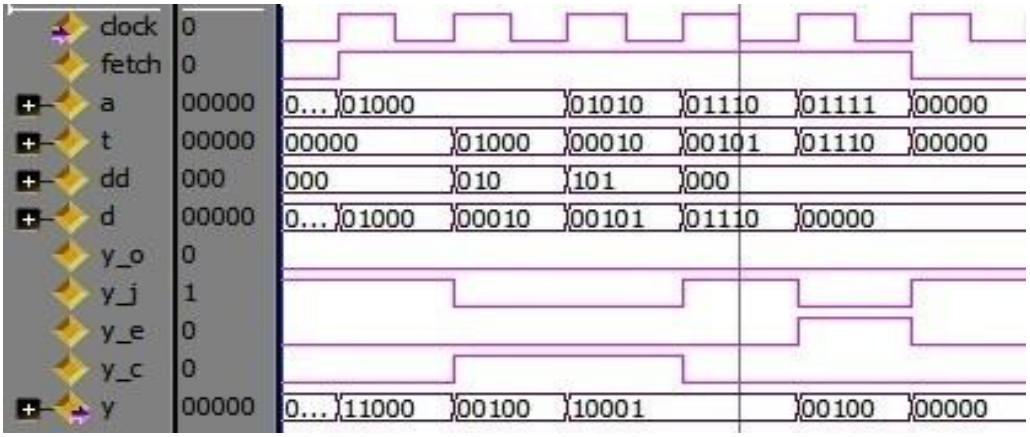

Figure 7. Simulation Waveform of CMCU with E-DIA without Idle Cycle

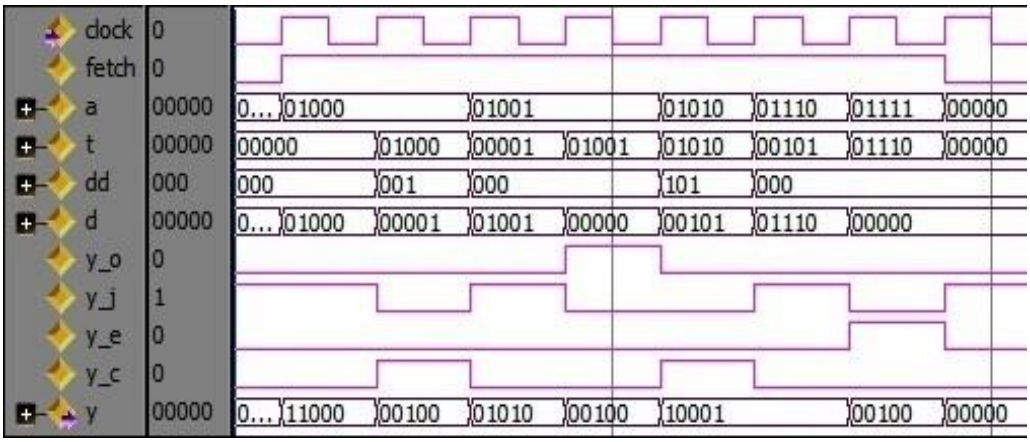

Figure 8. Simulation Waveform of CMCU with E-DIA without Microoperation Missing

In Figure 7 proceeding of $b_1 \rightarrow b_2 \rightarrow b_5 \rightarrow b_{10} \rightarrow b_{11}$ of FCA2 without an idle cycle is validated, and in Figure 8 proceeding of $b_1 \rightarrow b_2 \rightarrow b_3 \rightarrow b_4 \rightarrow b_5 \rightarrow b_{10} \rightarrow b_{11}$ of FCA2 without microoperation missing is validated.

4. Conclusions

The CMCU and CMCU with DIA were reviewed, and then the concept of TMI field of DIA was extended to e-DIA by extending TMI code assignment and coordinating the contents of corresponding AMP location. By applying e-DIA to CMCU, the occurrence of idle cycles was removed and micrpoperation missing or duplication in some cases was protected. The improvements from the removal of idle cycle and protection of microoperation missing or duplication were validated through simulation of VHDL implementation of CMCU with e-DIA.

Appendix

```
library ieee;
use ieee.numeric_bit.all;
entity CMCU_e_DIA is
  port(start, clock, x1, x2, x3, x4: in bit;
       y : out unsigned(1 to 5));
end;
architecture CMCU_e_DIA of CMCU_e_DIA is
signal t: unsigned(1 to 5) :="00000";
signal d: unsigned(1 to 5) :="00000";
signal a: unsigned(1 to 5) :="00000";
signal dd: unsigned(3 to 5) :="000";
signal m: unsigned(1 to 7) :="0000000";
signal f: unsigned(1 to 13) :="0000000000000";
signal fetch, y_j, y_o, y_c, y_e: bit :='0';
type cm_prom_1 is array(0 to 7) of unsigned(1 to 5);
constant cm1: cm_prom_1 :=
  ("01000", "01001", "01010", "01011", "01101", "01110", "01111", "10000");
type cm_prom_2 is array(0 to 31) of unsigned(1 to 7);
constant cm2: cm_prom_2 :=
  ("0011000", "0001010", "1010001", "0011000", "0000100", "0010001", "0001000", "0000110",
   "1000100", "0100100", "1010001", "0101001", "0100100", "1001010", "1100100", "1100010",
   "0101000", "0101010", "1000101", "0000000", "0000000", "0000000", "0000000", "0000000",
   "0000000", "0000000", "0000000", "0000000", "0000000", "0000000", "0000000", "0000000");
begin
  process(t, x1, x2, x3, x4)
  begin
    f <= "0000000000000";
    case t is
      when "01000" =>
          if x1='1' then f(1)<='1';
            elsif x2='1' then f(2)<='1';
            elsif x3='1' then f(3)<='1';
            else f(4)<='1';
          end if;
        when "00010" | "01010" =>
          if x2='1' and x3='1' then f(5)<='1';
            elsif x2='1' and x3='0' then f(6)<='1';
            elsif x2='0' and x4='1' then f(7)<='1';
            elsif x2='0' and x4='0' then f(8)<='1';
          end if;
        when "01101"=>
          if x2='1' and x3='1' then f(9)<='1';
            elsif x2='1' and x3='0' then f(10)<='1';
            elsif x2='0' and x4='1' then f(11)<='1';
            elsif x2='0' and x4='0' then f(12)<='1';
          end if;
        when "10010" =>
          f(13)<='1';
        when others =>
                    f <= "0000000000000";
    end case;
  end process;
    dd(3) <= f(4) or f(5) or f(6) or f(7) or f(8) or f(9) or f(10) or f(11) or f(12) or f(13);
    dd(4) <= f(2) or f(3) or f(6) or f(8) or f(10) or f(12);
    dd(5) <= f(1) or f(3) or f(5) or f(8) or f(9) or f(12);
    d(1) <= (a(1) and y_j) or ('0' and y_c);
    d(2) <= (a(2) and y_j) or ('0' and y_c);
    d(3) <= (a(3) and y_j) or (dd(3) and y_c);
    d(4) <= (a(4) and y_j) or (dd(4) and y_c);
    d(5) <= (a(5) and y_j) or (dd(5) and y_c);
    y_j <= not m(1) and not m(2);
    y_o <= not m(1) and m(2);
    y_c <= m(1) and not m(2);
```

```
    y_e <= m(1) and m(2);
  process(start, clock)
  begin
    if start='1' then t <="00000";
      elsif (clock'event and clock='1') then
        if y_o='1' then t <= t+1;
              else t <= d;
          end if;
    end if;
  end process;
 process(start, y_e, clock)
  begin
    if (clock'event and clock='1') then
        if start='1' then fetch <= '1';
                elsif y_e='1' then fetch <= '0';
          end if;
    end if;
  end process;
 process(fetch, t(3 to 5))
  begin
    if fetch='1' then a <= cm1(to_integer(t(3 to 5)));
              else a <= "00000";
    end if;
  end process;
 process(fetch, t)
  begin
    if fetch='1' then m <= cm2(to_integer(t));
              else m <= "0000000";
    end if;
  end process;
   y <= m(3 to 7);
end architecture;
```

Acknowledgments.

This work was supported by the intramural research grant of Chungbuk National University in 2015. The authors would like to express thanks to Prof. A. Barkalov, Prof. L. Titarenko and Prof. J. Bieganowski for their work on which this paper is based.

References

[1] A. Barkalov and L. Titarenko, "Logic Synthesis for FSM-Based Control Units", Springer, Berlin, **(2009)**.
[2] A. Barkalov and L. Titarenko, "Logic Synthesis for Compositional Microprogram Control Units", Springer, Berlin, **(2008)**.
[3] A.A. Barkalov, L.A. Titarenk and J. Bieganowski, "Synthesis of control unit with modified microinstructions", In Proc. of the Inter. Conf. Mixed Design of Integrated Circuits and Systems-MIXDES, **(2007),** pp. 157-160.
[4] A.A. Barkalov, L.A. Titarenk and J. Bieganowski, "Synthesis of control unit with modified system of microinstructions", In Proceedings of IEEE East-West Design & Test Symposium-EWDTS '07, (2007), pp. 545-549.
[5] V.A. Pedroni, "Circuit Design and Simulation with VHDL", 2nd edition, Cambridge, MA: MIT Press, **(2010)**.
[6] Altera Corporation, http://www.altera.com/
[7] http://bandi.chungbuk.ac.kr/~gysong/

