Andrei KARATKEVICH

University of Zielona Góra, Institute of Computer Engineering and Electronics

New Methods of Constructing Test Sequences for Datapath

Abstract. *The paper presents the original algorithms for generation of sequences of microinstructions for testing a datapath in a microprogrammed digital system. It is supposed that every direct connection between the datapath units should be tested, and the length of test sequence should be minimized. The proposed algorithms have been compared using the examples.*

Streszczenie*. W artykule przedstawiono nowe algorytmy generowania sekwencji mikroinstrukcji dla testowania ścieżki przetwarzania danych w mikroprogramowanym układzie cyfrowym. Zakłada się, że każde możliwe połączenie bloków ścieżki przetwarzania danych należy sprawdzić. Sekwencja testowa powinna być minimalizowana względem długości. Zaproponowane algorytmy porównano przy pomocy przykładów. (****Nowe metody konstruowania sekwencji testowych dla ścieżki przetwarzania danych****).*

Keywords: datapath, testbench, verification, microprogrammed system.
Słowa kluczowe: ścieżka przetwarzania danych, środowisko testowe, weryfikacja, układ mikroprogramowany.

Introduction: testing of datapath

A digital system, such as a processor or a controller, usually can be considered as a composition of a datapath and a control unit (fig. 1). For every component there exist specific methods of synthesis and verification [1-4].

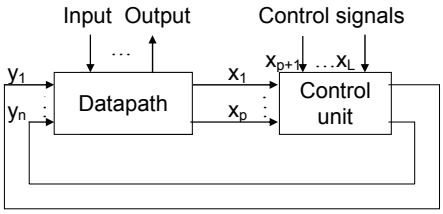

Fig.1. Digital system as a composition of control unit and datapath

For verification of a digital system project, appropriate testbench is usually created. Sequence of input values is a necessary component of a testbench. If the system under test is a datapath, there should be two kinds of inputs: signals from the external world and the microoperations from the control unit. Then the testbench should contain a sequence of signals from the control unit.

Specific features of testing of datapath of a microprogrammed system are the following [1]: every direct connection between the data path units should be tested; data can be directly written only to the input datapath units and directly read only from the output units; sequence of signals from the control unit consists of the microinstructions, where every microinstruction may contain several microoperations.

Example of datapath

We use as an example a design of a very simple processor intended to perform only two operations: bubble sort and finding maximal element in a table. It is an improved modification of the project described in [5]. The processor has been designed with the help of the experimental CAD system Abelite. Connection graph of the datapath for this processor is shown in figure 2.

Nodes of the connection graph correspond to the datapath units. We draw nodes without hatching to denote internal units, i.e. the units not available for reading and writing from outside; hatched nodes correspond to input and output units. Arcs correspond to possible direct data transfers between the units. Each arc is labeled with the microinstruction corresponding to the transfer. Loops correspond to the microoperations executed in one operational unit (such as $i:=i+1$). Multiple arcs describe a case in which different microinstructions send data between the same units.

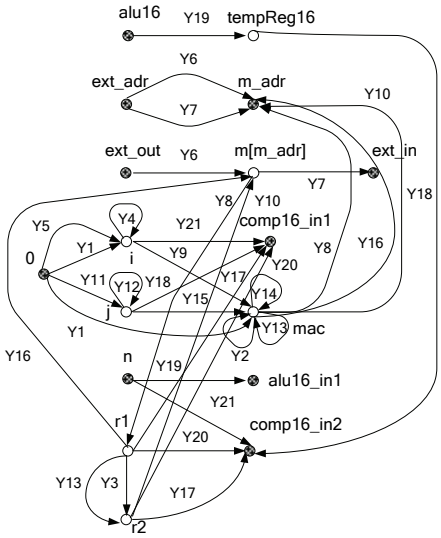

Fig.2. Connection graph

Generation of test sequences

The task considered in this paper can be solved by hand (as, for example, the test sequence presented in [1] was constructed). But for bigger designs it can be successfully done only with the help of computers, and the algorithms of test sequence generation turn to be necessary. Such algorithms, used by the companies producing CAD systems, are virtually not available in open publications.

Three algorithms for such generation have been developed by the author in cooperation with S. Baranov [6]. They are intended to construct minimized test sequences covering all direct connections in a datapath. The rest of this paper contains brief description and comparison of these algorithms. We will reference to them as to methods I, II and III, correspondingly (according to the order of their presentation in the paper).

I. Graph-based method

The first of proposed algorithms [6] is based on the observation that problem of covering of all arcs of a connection graph can be reduced, by contraction of all its sources and targets, to the so-called Chinese postman problem [7]. The problem is to find for given graph a shortest closed path that visits every arc. It can be solved in

polynomial time, and in such way the shortest sequence of microopeartions covering every connection can be quickly obtained.

But the test sequence should consist of microinstructions, not microoperations. A method is proposed in [6] of obtaining a sequence of microinstructions from a set of paths leading from input to output datapath units. Such paths can be obtained by partitioning of the "Chinese postman" path. The method is based on covering of microoperations by microinstructions in a way which avoids attempts of reading from the units to which no data have been written before (for details see [6]).

The shortest cycle covering all arcs in the graph obtained from the connection graph shown in figure 2 by merging its sources and targets (where the new node is denoted as v_0) is shown below:

$$v_0 \xrightarrow{Y6} v_0 \xrightarrow{Y7} v_0 \xrightarrow{Y19} v_0 \xrightarrow{Y21} v_0 \xrightarrow{Y19} tempReg16 \xrightarrow{Y18} v_0 \xrightarrow{Y6} m[m\_adr] \xrightarrow{Y7} v_0 \xrightarrow{Y6} m[m\_adr] \xrightarrow{Y8} r1 \xrightarrow{Y17} v_0 \xrightarrow{Y6} m[m\_adr] \xrightarrow{Y8} r1 \xrightarrow{Y20} v_0 \xrightarrow{Y6} m[m\_adr] \xrightarrow{Y8} r1 \xrightarrow{Y3} r2 \xrightarrow{Y20} v_0 \xrightarrow{Y6} m[m\_adr] \xrightarrow{Y8} r1 \xrightarrow{Y13} r2 \xrightarrow{Y10} m[m\_adr] \xrightarrow{Y8} r1 \xrightarrow{Y16} m[m\_adr] \xrightarrow{Y8} r1 \xrightarrow{Y3} r2 \xrightarrow{Y17} v_0 \xrightarrow{Y1} i \xrightarrow{Y4} i \xrightarrow{Y21} v_0 \xrightarrow{Y5} i \xrightarrow{Y9} mac \xrightarrow{Y2} mac \xrightarrow{Y8} v_0. \xrightarrow{Y11} j \xrightarrow{Y15} mac \xrightarrow{Y14} mac \xrightarrow{Y16} v_0 \xrightarrow{Y11} j \xrightarrow{Y12} j \xrightarrow{Y18} v_0 \xrightarrow{Y1} mac \xrightarrow{Y13} mac \xrightarrow{Y10} v_0$$

Applying to this path the method described in [6], we obtain the following test sequence:

Y6 Y7 Y19 Y1 Y4 Y21 Y5 Y9 Y2 Y8 Y20 Y13 Y10 Y8 Y11 Y15 Y14 Y16 Y8 Y3 Y17 Y12 Y18

II. Petri net-based method

As far as single microinstruction may perform several different connections between datapath units, a Petri net [8] seems to be more adequate datapath model, then a connection graph. We can model datapath units as Petri net places, and microinstructions as transitions. An algorithm has been developed [9] which generates test sequence using such model.

Let us construct a Petri net for given datapath, where every transition has input places corresponding to the units, from which the microinstruction labeling this transition reads some data, and output places corresponding to the units to which the microinstruction writes. Next all the places corresponding to input and output units of the datapath are removed from the net. For every T-invariant of the obtained net, such that all its components are positive, exists a firing sequence, which leads from empty marking back to empty marking and contains every transition at least once (it follows from Theorem 34 from [8]). A sequence of microoperations corresponding to such firing sequence contains every microoperation at least once and moves data from input to output datapath units without lost of data and without attempts to read from the internal units to which no data were written before [9].

A minimal positive T-invariant can be obtained as a corresponding solution of the system of linear equations describing structure of the net [8]. Having such solution, it is possible to obtain the test sequence. Then the sequence can be minimized by removing from it the repeating microinstructions which do not write new data to the units (under the assumption that the data in the input blocks remain unchanged).

T-invariants for our example are the integer solutions of the system of equations (1) (where numbers of the variables correspond to the numbers of microinstructions):

$$(1)\quad \begin{cases} -x_{18} + x_{19} = 0 \\ x_6 - x_7 - x_8 + x_{10} + x_{16} = 0 \\ x_1 + x_5 - x_9 - x_{21} = 0 \\ x_{11} - x_{15} - x_{18} = 0 \\ x_1 - x_8 + x_9 - x_{10} + x_{15} - x_{16} = 0 \\ -x_3 + x_8 - x_{13} - x_{16} - x_{17} - x_{20} = 0 \\ x_3 - x_{10} + x_{13} - x_{17} - x_{20} = 0 \end{cases}$$

To obtain the solution of (1) which we need, it is necessary to minimize the function $f = x_2 + x_4 + 2x_7 + 2x_9 + 3x_{10} + x_{12} + x_{14} + 2x_{15} + 2x_{16} + 6x_{17} + 3x_{19} + 6x_{20} + 2x_{21}$, where all variables are required to be positive integers. Such solution, being the T-invariant we are looking for, is given by the vector (3,1,2,1,2,5,1,6,4,1,2,1,1,1,1,1,1,1,1,1,1). The following firing sequence can be obtained from it: $t_1t_2t_4t_6t_7t_6t_8t_3t_6t_9t_5t_8t_6t_9t_5t_{11}t_{12}t_{14}t_{17}t_8t_3t_6t_{15}t_8t_9t_{10}t_1t_{11}t_{13}t_8t_9t_{16}t_1t_8t_{19}t_{18}t_{20}t_{21}$.

The corresponding sequence of microoperations after minimization is presented below:

Y1 Y2 Y4 Y6 Y7 Y8 Y3 Y9 Y5 Y8 Y11 Y12 Y14 Y17 Y8 Y15 Y10 Y13 Y16 Y8 Y19 Y18 Y20 Y21.

III. Heuristic method

In both methods described so far a "long" sequence is constructed at first, and then it is minimized. A question arises: is it possible to build a minimized sequence directly? It would allow obtaining a quicker algorithm, however not always providing optimal results. Such algorithm (never presented in English before) is described below.

Let $Y_i^\bullet$ denote the set of internal datapath units to which microinstruction Y_i writes data, and ${}^\bullet Y_i$ - the set of blocks from which Y_i reads. Below M denotes the set of all microinstructions, L is an initially empty list which will contain the test sequence, B is the set of internal datapath units. Every internal unit will be labeled by a "color" – white, grey or black. The algorithm consists of the following steps.

1. Initialize S as the set of all microinstructions which do not read from the internal units (${}^\bullet Y_i = \varnothing$). Assign white color to all internal units. $T := M \setminus \{Y_j \in M \mid {}^\bullet Y_i = Y_i^\bullet\}$; $Q := T$.
2. While $Q \neq \varnothing$ and not all blocks in B are black, do:
   a. If $S \cap Q \neq \varnothing$ then
      i. select any microinstruction $Y_i \in S \cap Q$;
      ii. add Y_i to L; $Q := Q \setminus \{Y_i\}$;
      iii. color every unit belonging to $Y_i^\bullet$ as grey;
      iv. color every unit belonging to ${}^\bullet Y_i$ as black;
      v. $S := S \setminus \{Y_j \in T \mid Y_i^\bullet \cap Y_j^\bullet \neq \varnothing\}$;

vi. $S := S \cup \{Y_j \in T \mid Y_i^{\bullet} \cap {}^{\bullet}Y_j \neq \emptyset$, every unit in ${}^{\bullet}Y_j$ is grey or black and no unit in ${}^{\bullet}Y_j$ is grey$\}$;

b. else

i. select any grey unit b;

ii. find shortest path in the connection graph from b to an output unit starting from an arc corresponding to a microinstruction $Y_i \in S$; add Y_i to L;

iii. perform for Y_i the operations described in 2.a.iii – 2.a.vi.;

3. For every microinstruction $Y_i \in M \setminus T$:

a. add Y_i to L after the microinstructions writing to every unit in ${}^{\bullet}Y_i$.

White color assigned to a unit means that no data have been written to this unit yet. Grey color means, that data were not read from this unit after last writing to it. Otherwise (if the data were read after the last writing) the unit has black color. When the algorithm stops all the units should be black (if a white unit would remain it would mean that not every connection is covered by the test; remaining of a grey unit would mean that some data were not propagated to an output). S is the set of microinstructions such that every of them can be added to the list L at current step (and it will not cause attempts of reading from the "white" units or writing to the "grey" ones). Q is the set of microinstructions which do not appear in L yet. If necessary, data from the "grey" units are moved forward to the outputs by means of the heuristic procedure described in item 2b of the algorithm. It is reasonable to consider the self-loop microinstructions separately; that is the reason why they are not added to Q at the beginning of the algorithm, and are added to the test sequence at item 3.

We illustrate the algorithm using the example of datapath shown in figure 2. At first all internal units are "white", and the set S consists of the microinstructions reading data only from the input units: $S = \{Y1, Y5, Y6, Y11, Y19\}$. For example, microoperation $Y19$ reads from the units **n** and **alu16**, which are the input units (the corresponding nodes of the graph from figure 2 have no incoming arcs).

Now we can select any microinstruction from S and add it to the sequence. Let us select for example $Y1$ (now $L = (Y1)$). $Y1$ writes to the units **i** and **mac**, and now those units are labeled as grey (step 2.a.iii of the algorithm). Microoperation $Y1$ is removed from the sets Q (2.a.iii) and S (2.a.v); microoperation $Y2$ is added to S. Now $S = \{Y6, Y11, Y19, Y21\}$; let us select $Y6$. Next **m[m_adr]** is colored as black (it is the only internal unit to which $Y6$ writes), and $S = \{Y7, Y8, Y11, Y19, Y21\}$. $Y8$ is added to S because this microinstruction reads from the units **mac** and **m[m_adr]**, which are grey at this moment. At the next step we may select $Y11$ ($L = (Y1, Y6, Y11)$), then $Y7$ (now unit **m[m_adr]** is the first unit colored as black, because $Y7$ reads from it).

The operations of step 2a of the algorithm should be executed while the set $S \cap Q$ is not empty. When it becomes empty, $L = (Y1, Y6, Y11, Y7, Y8, Y3, Y9, Y5, Y17, Y13, Y19, Y18, Y20, Y21, Y10, Y15)$, $Q = \{Y16\}$ (the only microinstruction which does not appear in the sequence yet), units **m[m_adr]** and **mac** are grey, all other internal units are black, $S = \{Y3, Y5, Y7, Y8, Y11, Y13, Y17, Y18, Y19, Y20, Y21\}$. The shortest path from **m[m_adr]** to a target of the connection graph leads through the arc labeled with $Y7$. When $Y7$ is added to L, **mac** remains the only grey node, and the microinstructions writing to **m[m_adr]** ($Y6$, $Y10$, $Y16$) are added to the set S. Now $S \cap Q = \{Y16\}$, and $Y16$ should be added to the sequence. When it is added, unit **mac** is colored as black, but unit **m[m_adr]** again becomes grey. Adding $Y7$ once more to the sequence allows to get rid of all grey nodes.

The sequence obtained when the main loop of the algorithm stops is the following:

$Y6\ Y11\ Y7\ Y8\ Y3\ Y9\ Y5\ Y17\ Y13\ Y19\ Y18\ Y20\ Y21\ Y10\ Y15\ Y7\ Y16\ Y7$.

After execution of step 3 of the algorithm (adding the self-loop microinstructions) the following sequence is obtained:

$Y1\ Y2\ Y4\ Y14\ Y6\ Y11\ Y12\ Y7\ Y8\ Y3\ Y9\ Y5\ Y17\ Y13\ Y19\ Y18\ Y20\ Y21\ Y10\ Y15\ Y7\ Y16\ Y7$.

Comparison of the methods

For the considered example methods I and III provide the sequences a little bit shorter than method II. But in fact method I does not guarantee neither forwarding of the data from the internal units to the outputs, nor lack of situations in which new data are written to an internal unit before reading from it the data written before. It is illustrated by the examples shown in figure 3. For the example a) method I calculates the sequence $Y1\ Y2\ Y4\ Y5\ Y2\ Y3$; after the second execution of $Y2$ data from unit **a** will not be moved to the output. For the example b) method I provides the sequence $Y1\ Y2\ Y3$ (there are two writings to unit **a** and only one reading from it). For c) it gives $Y1\ Y2\ Y3\ Y4\ Y5\ Y6$ (in this case microoperation $Y2$ has not been tested in fact, because $Y3$ destroys its results).

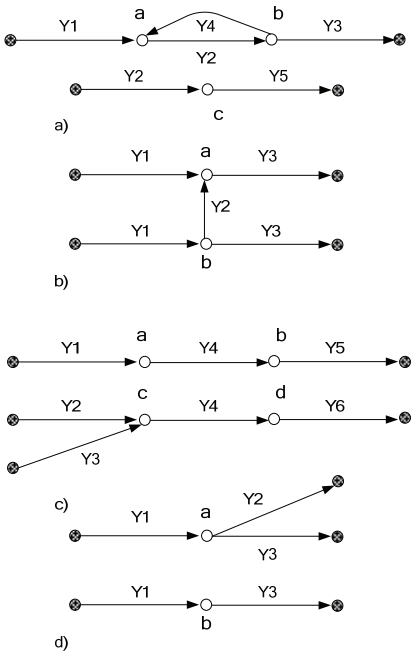

Fig.3. Counterexamples for methods I and II

The problems with method I are caused by its last step, obtaining test sequence from the set of paths. The method provides correct and optimal result when every microinstruction moves data only between two units. That means that it is reasonable to use this method when every connection in the datapath can be tested separately (i.e., sending data between any two units does not cause simultaneous sending between some other units).

Method II had been developed for dealing with the mentioned difficulties, but there are some other problems connected with it. For a Petri net modeling a datapath a T-invariant with all positive entries does not always exist. When it exists, the method provides correct result; for

example, this is the case for connection graphs presented in figures 3a and 3c ($Y1\ Y2\ Y3\ Y5\ Y4\ Y2\ Y3\ Y5$ and $Y1\ Y2\ Y4\ Y5\ Y6\ Y3\ Y4\ Y5\ Y6$, correspondingly). But for the examples shown in figures 3b and 3d systems of linear equations constructed by method II have no solutions meeting the specified conditions. However the correct test sequences for these examples do exist: $Y1\ Y3\ Y2\ Y3$ for 3b) and $Y1\ Y2\ Y3$ for 3d).

Method III seems to be comparatively the best one. It allows getting good solutions in many cases in which methods I and II fail (for example, in all cases presented in figure 3). It is possible however to construct a datapath for which the heuristic approach applied in method III will cause producing an extremely long test sequence or even entering an infinite loop.

Conclusion

Developing of the formal methods for test generation for the digital devices is evidently practically important. The proposed methods contribute to such developing in the field related to the datapaths. The first of methods seems to be promising for the systems which consist of control unit and datapath, but which are not microprogrammed. The second method is efficient for the microprogrammed systems but fails being applied to some of datapath structures. The third method is universal and, after some improvements, may turn to be the most efficient one.

I would like to thank Samary Baranov for cooperation, fruitful discussions and commentaries which helped to improve this paper.

REFERENCES

[1] Baranov S., *Logic and System Design of Digital Systems*. Tallinn: TGU (2008)
[2] Barkalov A., Węgrzyn M., *Design of control units with Programmable Logic*. Zielona Góra: UZ (2006)
[3] Barkalov A., Titarenko L., *Basic Principles of Logic Design*. Zielona Góra: UZ (2010)
[4] Wiśniewski R., *Synthesis of compositional microprogram control units for programmable devices*. Zielona Góra: UZ (2009)
[5] Baranov S., ASMs in high level synthesis of EDA tool ABELITE, *DESDes'09 Int. IFAC Workshop Proceedings*, (2009), 195-200 (available online on www.ifac-papersonline.net)
[6] Karatkevich A., Baranov S., Graph Based Approach to Test Bench Constructing for Datapath, *55. Internationales Wissenschaftliches Kolloquium - IWK,* (2010), 662-667
[7] Edmonds J., Johnson E.L., Matching Euler tours and the Chinese postman problem, *Mathematical Programming*, 5 (1973), 88-124
[8] Murata T., Petri Nets: Properties, Analysis and Applications, *Proceedings of the IEEE*, 77 (1989), 541–580
[9] Karatkevich A., Petri Net Based Approach to Test Bench Constructing for Datapath, *1st International Conference PECCS Proceedings*, (2011), 506-511

***Author**: dr hab. Inż. Andrei Karatkevich, University of Zielona Góra, Institute of Computer Science and Electronics, ul. prof. Z. Szafrana 2, 65-246 Zielona Góra, E-mail: A.Karatkevich@iie.uz.zgora.pl*