UDC 004.3

D.S.Tyanev
Technical University of Varna, Bulgaria
dstyanev@yahoo.com

Non-Linear Asynchronous Micro-Pipelines

The paper considers structural problems in synthesis of micro-pipelines which implement algorithms with conditional jumps. These structures require pre-definition of the term "micro-pipeline". As a result there are defined, analyzed and described four new scientific tasks necessary for solving this common problem. The paper presents the decision of only one of the tasks – synthesis of micro-pipeline that controls section generating value of the transition condition, as well as the connection of this section with initial stage automates into both branches. The complete logical synthesis is explained and as a result logical structures of pipeline controllers are obtained in two variants: for 2-phase transfer protocol controller and for 4-phase data transfer protocol.

***Key words**: Micro-pipelines, Branches, Data Transfer Protocols*

Introduction

The paper considers the structure of presented on Figure 1 model algorithm, which is implemented and whose execution has a pipeline organization. It is assumed that this algorithm is detailed and its realizable blocks are implemented through the methods, discussed in [8], [9], [10], [12] and others. Each realizable block from the block-diagram is a particular micro-pipeline stage (one- or multi-cycle) according to definitions in [11] and [13]. It means that each realizable block can be considered as a multi-cycle stage in terms of the possible operations when interpreting the algorithm. Our understanding is that each multi-cycle stage can have more complicated internal structure similar to the presented, composed of random micro-pipeline stages. As a control methods for micro-pipeline stages can be considered either synchronous or asynchronous methods, as well as any combination of them.

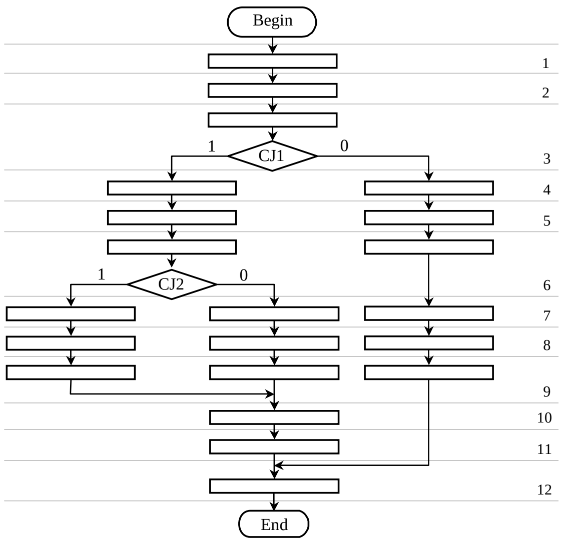


Fig. 1. Exemplary algorithmic structure

As it is seen, presented algorithm contains few linear sections, but generally can be defined as branch, despite the above considerations. Both branch conditions *CJ* (conditional jump) form the possible computational paths as follows:

1. *Begin*; 1; 2; 3 (CJ1=true); 4; 5; 6 (CJ2=true); 7; 8; 9; 10; 11; 12; *End* .
2. *Begin*; 1; 2; 3 (CJ1=true); 4; 5; 6 (CJ2=false); 7; 8; 9; 10; 11; 12; *End* .
3. *Begin*; 1; 2; 3 (CJ1=false); 4; 5; 6; 7; 8; 9; 12; *End* .

where 1,2,3,... denote number of the stages execution is crossing.

Assuming that every *Begin-End* path is unique, the corresponding micro-pipeline stages into the parallel branches are located at one and the same serial level of the micro-pipeline, which are 12 in this case. At levels from 4 to 9, where are several micro-pipeline stages, at the moment of every single execution works only the stage included into current algorithmic branch.

Term "micro-pipeline"

The problem for hardware implementation of algorithm with conditional jumps requires new understanding of the term "micro-pipeline", which will be described.

With pipeline organization, data links between pipeline registers are managed by pipeline controllers. It means that switching of each controller at level with branches depends on the value of corresponding condition for transition. As micro-pipeline extension, after level producing condition for transition, the implementation of the both algorithmic branches is required. So the term micro-pipeline is formally broken because is broken common understanding of sequential order of the micro-pipeline stages. But in presented case the physical presence of all possible computational branches

Begin-End is inevitable and is a result from community property of every algorithm. Despite of the number of possible computational branches for certain problem, each path is unique as sequential passes through consecutive activated micro-pipeline stages and in this meaning these stages create a chain, which corresponds to the term micro-pipeline. Therefore, the presence of different implemented parallel branches does not contradict to the general understanding about micro-pipeline and each similar structure can be defined as micro-pipeline. In other words, it can be assumed that the structure consists of several pipelines with united common parts in way to have one beginning and one end.

New aspects

Micro-pipeline implementation of such type common algorithmic structures meets new and versatile problem – computational process control in alternative conditions. Problem's analysis presents few new and unresolved aspects. Analysis and definition of these new aspects, related to the general formulation accepted, we assume as an independent result from our research.

1. At first place, the obvious aspect of the problem is the synthesis of pipeline controller at the point of conditional jump. This controller differs significantly from the ordinary linear state machine because it must choose one of the two algorithmic branches. This aspect is inevitable connected with the next. All of the new aspects are related with design of pipeline controllers at specific points of the common algorithmic structures.

Original pipeline controllers managing the transfer between one- and multi-cycle and mixed types of micro-pipeline stages we consider in [11], [13] and others. The common part between them is that these controllers with all their variety support only linear micro-pipelines, exactly as most of the considered in public structures, starting with fundamental [14]. Implementation of micro-pipeline stage with conditional jump we have discussed in [8], where is presented the problem's analysis and are proposed two variants of its realization. They coincide in the fact that value of the logical condition *CJ* is used to control data bus in order to direct the results to the current branch, implementing particular algorithmic path *Begin-End*. The last corresponds to levels 4 and 7 in the diagram at Figure 1 (pay attention to the lines, limiting the stages). Defined task in [8] for data transfer control into branched micro-pipeline is also new and its solution will be presented in this paper. The essence of this task is the synthesis of pipeline controller to manage the micro-pipeline stage generating transition condition. The logical value of the condition must define in which branch will go *Request*. This is related with problem for receiving the signal *Acknowledgement* from the stage already received the request. Because of the unbreakable nature of signals *Req* and *Ack* we assume it as one task, despite of the two particular decisions it has.

2. At second place is the aspect for synthesis of pipeline controller about the stages at the common points of the algorithm. As it can be seen from Figure 1, at levels 9 and 12 the input points of corresponding micro-pipeline stages join more than one output from previous stages. The entry to the input of results from several previous stages, placed parallel topologically, present new and independent problem. The decision is to synthesize controller managing receiving stage with parallel in time entry of generally more than one *Request*. In the mirror-sense, this aspect is related with one more problem – the task for generating *Acknowledgement* signal to the corresponding previous stage. Because of the unbreakable nature of signals *Req* and *Ack* we assume it as one task, although it also has two particular decisions.

3. The incoming to the receiving stage at the common point requests from parallel previous stages present the third aspect of the problem – the request choice. To solve the request choice and to receive the corresponding data, the receiving stage pipeline automat must execute arbiter procedure. The realization of this procedure presents the third aspect. Requests arbitration is well-known and there are different implementations [15], but we consider it as a new one in terms of micro-pipeline control.

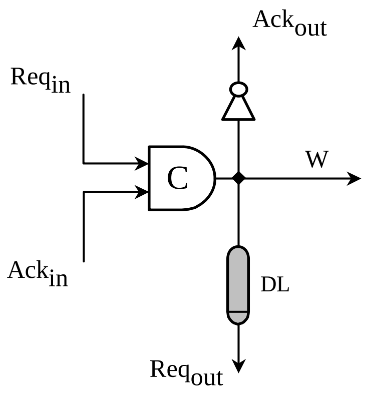
4. And finally the fourth aspect: at the joint point, where several branches are united, the requests attended with obtained data refer to different tasks, started into the micro-pipeline. The order in which the results are coming to the joint point is not definitely the same as the order in which were started the corresponding tasks. In other words, at receiving stage containing the joint point the data will not come in right order. So the presence of branches in the micro-pipeline leads to problem: the pipeline's outgoing final results barely will be in the order, corresponding to the starting one. Obviously, a new problem must be defined, fourth in a row, which requires introduction of order and accordance identification system for the final results. The problem for order restore is known. At the processor pipelines level this problem has software decision. This is the reason to assume that working conditions for processor pipelines can not be compared with that for the micro-pipelines considered in this paper.

The problem for implementation of common algorithmic structure obviously is the topic of the day. In publications [1], [2], [3], [4], [5], [6], [7] can be found decisions which are particular elements of the considered problem. There are no formulated and analyzed micro-pipelines with common structure in their integrity.

In this paper we present solutions only for the first aspect from section 2, referring to the both types of transfer protocol.

Micro-pipeline with two-phase transfer protocol

As it was mentioned, the pipeline controllers are responsible for computations management so the essence of the first part of the problem is to design dependent on the conditional jump pipeline controller. As a first variant for decision we consider controller using asynchronous Mueller C-element. Such controllers (Figure 2) implement 2-phase transfer protocol. Events in this protocol – request Req_{in}, acknowledgement Ack_{in} and micro-pipeline stage functioning *Work* for two consecutive work cycles *(k)* and *(k+1)* – are numbered in the protocol time-diagram as follows:



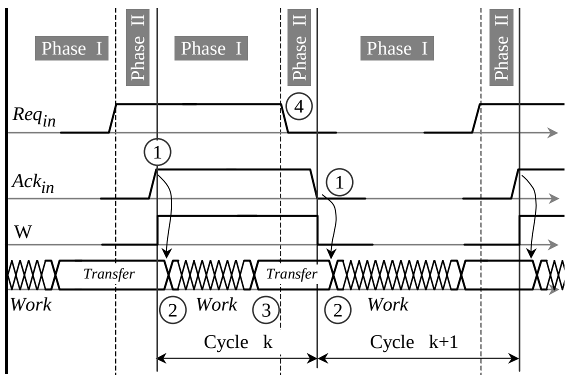


Fig. 2. Two-phase controller and protocol

1. Signal Ack_{in} is switched to 1. When $Req_{in}=Ack_{in}=1$, the controller of current stage is switched to 1 (state 1), generating write data signal for its fixing register (interval *Transfer*) ;
2. In the logic after the register with new written data starts transitional process – beginning of computations into current stage (interval *Work*) ;
3. Computations are finished – the result is ready ;
4. In this condition, the controller of current stage waits the switching of Req_{in} from the previous stage into low level (Req_{in}=0), as well as acknowledgement for data receive in the next stage (Ack_{in}=0) ;

Next transfer cycle (k+1) starts with the same actions:

1. When $Req_{in}=Ack_{in}=0$, new cycle begins. The pipeline controller of current stage switches back to low level (state 0), generating signal for data write into its register ;
2. Data write is done and into current stage starts new computation cycle.

The time-diagram allows to consider that the four possible combinations of input signals values (Req_{in}, Ack_{in}) are equally divided between the two states of pipeline automat – combinations 11, 01 for state 1, and combinations 00, 10 for state 0.

Figure 3 shows part of the micro-pipeline structure consisting stage, which generates conditional jump *CJ*. There are two additional logical schemes to the logical structure of the managing current stage controller, which synthesis will be presented later. These two additional schemes are *LA* – scheme, setting up the signal *AckCJ*, and *LR* – scheme, generating actual requests Req_{true} и Req_{false}.

The structure presents also the input registers of the both algorithmic branches. Whoever branch will be chosen for current computations, should start that initial state machine which is defined by the actual value of logical condition *CJ*. In fact, there is a fixing register *RG_F* in the beginning of each branch, but micro-operation write after signal *W* (*Write*) should be executed only in one of these registers.

Acknowledgement to conditional jump controller

The switching of pipeline controller into stage with conditional jump is a function of two signals: Ack_{true} and Ack_{false}, indicating readiness of each branch independently, i.e. they are "parents" of the signal *AckCJ*. Toward the stage with conditional jump, signals Ack_{true} and Ack_{false} have the same parity, but in the time they are competitive. The last means that in the time their switching moments can be either same or opposite in view of its logical value. Pay attention that the competitiveness between these switching is insignificant assuming transit condition *CJ*. *CJ* value is present at the same time with obtained result and request *ReqCJ*, which must reach the controller into chosen branch. As response will come the acknowledgement we are talking over (Ack_{true} or Ack_{false}, depends on *CJ*).

In other words, only the acknowledgement from the branch, received the request, must be admitted to the conditional jump automat. This is possible because the stage still sustains the current value of *CJ*. Only the logical value of acknowledgement has to be considered. Taking into account that the controller of conditional jump stage attends two branches and pre-history of its switching is not known, there is no guarantee that the logical value of the returned acknowledgement will be correct. Therefore the current condition of the automat must be considered as well. For example, if it is in state 1, the next switching can be only to 0. This switching is possible only if the acknowledgement is zero. Controller of sending stage is always connected to the controller of receiving stage in the time of transfer (connection is supported by *CJ* value), so the above considerations can be presented as follows:

$$AckCJ = Ack_{true} \cap CJ \cup \cup Ack_{false} \cap \overline{CJ} \ . \qquad (1)$$

The acknowledgement *AckCJ* will have the value of the incoming acknowledgement from corresponding branch so there will be no need to invert of that value. Equation (1) will implement the LA-scheme (Figure 3).

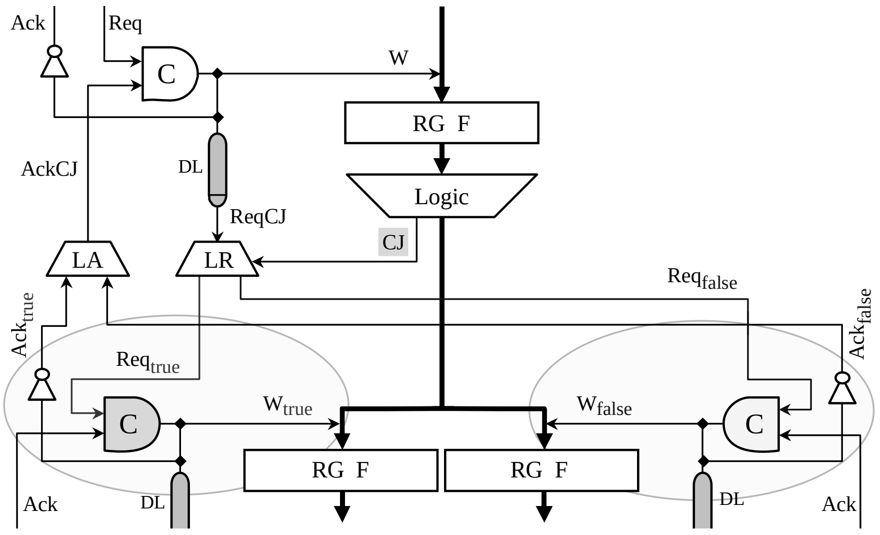


Fig. 3. Structure of stage with condition jump

Branch requests generation

The pipeline automat controlling the conditional jump stage propagates to the next stages request, denoted as *ReqCJ*. Unfortunately, this request cannot be lead directly to the controllers' inputs at the beginning of each alternative branch. Corresponding inquiries, which the state machines must receive, are denoted as Req_{true} for the branch "*true*" and Req_{false} for the "*false*"-branch, and are function of the LR-scheme (Figure 3). The direct inclusion of *ReqCJ* is not possible because in the branching point it enters in complicate functional connection with the logical value of *CJ* on one hand and with the current state of the automat at the beginning of each branch on other hand. This is a consequence of the types of automats, which use 2-phase protocol for transfer control. The last means that each their switching (0→1 and 1→0) causes "*write*" into the fixing registers and starts the stage computations. If we consider switching in the beginning of a micro-pipeline branch, except of the *CJ* logical value (0 or 1) must be taken in consideration also the initial condition of corresponding controller. Pay attention that the C-element is switching with two initial ones as well as two zeros. It means that the signals Req_{true} and Req_{false} are functions not only of *ReqCJ* switching and *CJ* value, but also of the automats' conditions at the beginning of branches. For example, if the value of transition condition is one (*CJ*=1), it means that computations must continue into the "*true*"-branch (Figure 3). If the state of pipeline controller in this branch is one, supported in time from Req_{true}=1, it must be switched in 0-state to start these computations by falling edge of the signal W_{true}. For this purpose, at input of this C-element must be two zeros combined. Do not forget that each new value of *ReqCJ* (either 0 or 1) presents new request from the controller to the branch stage.

The above logic is expressed by the following truth-tables, where signals W_{true} and W_{false} present the state of corresponding C-elements.

Based on the above truth-tables, following logical functions are synthesized:

$$Req_{true} = CJ \oplus W_{true} \ . \quad (2)$$

$$Req_{false} = \overline{CJ \oplus W_{false}} \ . \quad (3)$$

As it shown, the logic of Req_{true} and Req_{false} does not depend on *ReqCJ*, as it was expected. Dependency of Req_{true} and Req_{false} is not on the *ReqCJ* value but on the time, i.e. on the switching moments of *ReqCJ*. It means that the change in *ReqCJ* value, i.e. edge appearance, indicates the moment in time when the stage logic finishes its computations. This moment does not compulsory coincide with the appearance of the *CJ* true value. Depending on the *CJ* computation complexity, in general case should be assumed that the *CJ* true value can appear earlier than the new edge of *ReqCJ* or at least at the same time and never later than it. With direct implementation of (2) and (3), earlier appearance of *CJ* will lead to earlier creation of Req_{true} and Req_{false}, which on the other hand will start earlier the corresponding pipeline branch. This beginning will start with data writing into fixing register but the data still won't be reached in time their true values. In this way, it is possible the computation to be started with wrong data.

The main conclusion of the above considerations is that the (2) and (3) equations define request values, but the moment when they will appear and start to affect is specified by *ReqCJ* switching moment. In other words, Req_{true} and Req_{false} new values must appear as a response of *ReqCJ* edge. This means that creation of Req_{true} and Req_{false}, is not possible only with combinational logic.

Table 1 Request to controller in "*true*"-branch

CJ	W_{true}	ReqCJ	Req_{true}
0	0	Falling edge appearance ↘0	0, no switching
0	0	Rising edge appearance ↗1	0, no switching
0	1	Falling edge appearance ↘0	1, no switching
0	1	Rising edge appearance ↗1	1, no switching
1	0	Falling edge appearance ↘0	↗1 , switching
1	0	Rising edge appearance ↗1	↗1 , switching
1	1	Falling edge appearance ↘0	↘0 , switching
1	1	Rising edge appearance ↗1	↘0 , switching

Table 2 Request to controller in "*false*"-branch

CJ	W_{false}	ReqCJ	Req_{false}
0	0	Falling edge appearance ↘0	↗1 , switching
0	0	Rising edge appearance ↗1	↗1 , switching
0	1	Falling edge appearance ↘0	↘0 , switching
0	1	Rising edge appearance ↗1	↘0 , switching
1	0	Falling edge appearance ↘0	0, no switching
1	0	Rising edge appearance ↗1	0, no switching
1	1	Falling edge appearance ↘0	1, no switching
1	1	Rising edge appearance ↗1	1, no switching

Presented considerations prove that time dependency of computed values (2) and (3) can be implemented only by storage element – flip-flop. Because the "*write*" must be done on each C-element switching, the synchronizing flip-flop must be DEDTFF (D flip-flop working on both edges). At the final logical structure of pipeline controller (Figure 4) is presented our preferable decision, based on typical D-Latch flip-flop and two edge-detectors – FD↑ for rising edge and FD↓ for falling. LR-scheme (Figure 3) contains the C-element of conditional jump branch; LA-scheme, joining acknowledgements Ack_{true} and Ack_{false}, as well as both schemes generating Req_{true} and Req_{false}.

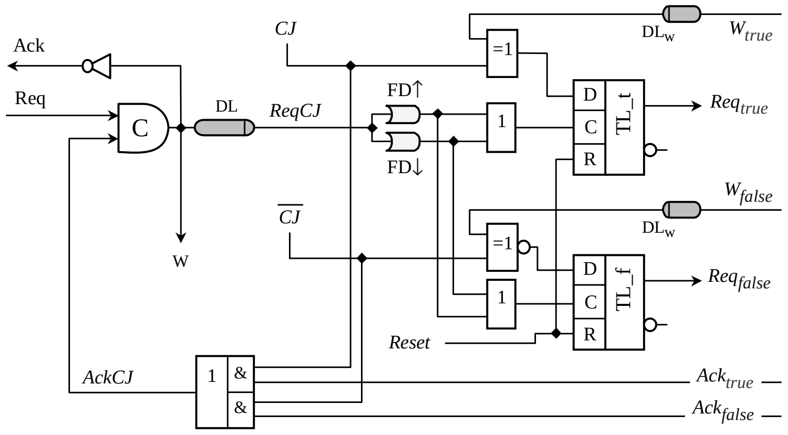


Fig. 4. Logic scheme of 2-phase pipeline controller into conditional jump stage

As it is shown on the logic structure, pulse generators are joining into OR-element and realize “*write*” on flip-flop C-input. Obtained by (1) value of Req_{true} comes on D-input and is stored into flip-flop until the next time when the same branch will be chosen. The *Reset* signal is necessary in the beginning when all pipeline controllers are forced to initial state. Similar scheme creates Req_{false} to the controller into “*false*”-branch, depending on (2).

In response of Req_{true} or Req_{false} corresponding pipeline automat will be switched and will turn through feedback new value of signal *W*, which threaten reliability of “*write*” to TL_t or TL_f, so we must hold in time the value until the “*write*”-impulse disappears from C-input. The delay is provided by DL_W.

Micro-pipeline with 4-phase transfer protocol

Second decision is about pipeline controllers, implementing 4-phase transfer protocol. Figure 5 presents one of synthesized in [11] controllers, realizing 4-phase protocol with anticipating reset. The events into protocol for two consecutive work cycles (k) and (k+1) of particular micro-pipeline stage are presented on the time-diagram and are numbered as follows:

1. Micro-pipeline stage is finished computations. Obtained result is received from the next stage. Current stage indicates this with signal Ack_{out} and pipeline controller is expecting signals Req_{in} and Ack_{in} ;
2. Both input ones Req_{in}=Ack_{in}=1 switch C-element to one (*W*=1) and new data is written to the fixing register of current stage. It starts transitional process – begins result computations;
3. C-element condition is stored into DE flip-flop with delay *DL1*. The delay provides the time necessary for writing in flip-flops of fixing register. After writing, the inverse input of DE flip-flop resets the C-element via feedback, setting it in this condition in advance, i.e. before the computations are over, preparing the C-element for the next cycle ;
4. Computations of the current stage are finished and the controller sends Req_{out} to the next stage.

New cycle is started into the current stage.

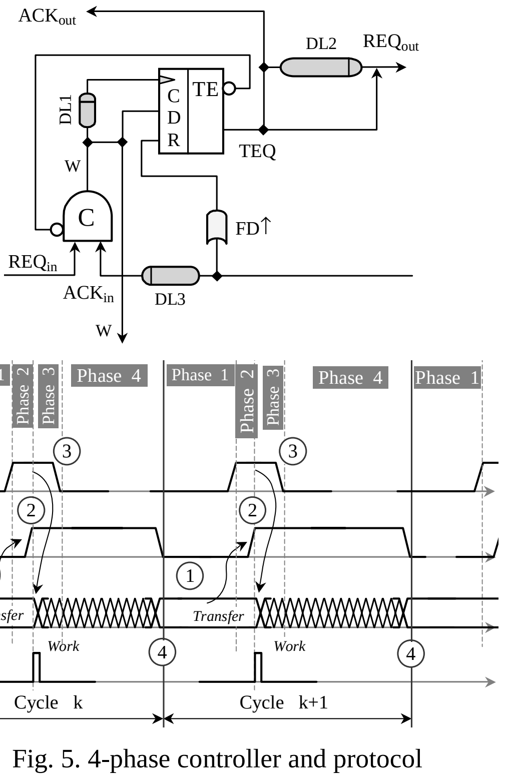


Fig. 5. 4-phase controller and protocol

Acknowledgement synthesis for conditional jump controller

Presented protocol shows that pipeline controllers, including from the beginning of each branch, are waiting their start into zero-condition. In other words, the conditions for starting these controllers are always one and the same unlike presented 2-phase controllers.

Switching of pipeline controller into conditional branch stage is a function of two signals: Ack_{true} и Ack_{false}, indicating the readiness of each branch. The logic analysis of events at conditional jump point is similar to that made for the previous controller. Both acknowledgement signals are also competitive in time. As requests have always the same value, the acknowledgements have same value too. The appearance of each signal (Figure 6, moment 2) can switch the controller without carrying about the presence of the other, which could be assumed as normal if it is certain that *CJ* (which will receive its value with delay) will have such value that the computations will continue into branch caused the start. But if it is not like this and computations must continue into the branch with delayed acknowledgment, already generated *ReqCJ* must wait for corresponding event. As the controller of sending stage is constantly connected to the controller of receiving stage in the time of transfer (connection is provided by *CJ* value), joining of acknowledgements from the both branches is achieved by logic disjunction. This statement is similar to already presented for the 2-phase controller so the logic of *AckCJ* is expressed by equation (1).

Request generation to the barnches

From its own part, *ReqCJ* does not depend on pipeline automat condition into branches. This means that requests Req_{true} and Req_{false} depend only on *CJ* value which leads to following statements:

$$Req_{true} = ReqCJ \cap CJ \ , \quad (4)$$

$$Req_{false} = ReqCJ \cap \overline{CJ} \ . \quad (5)$$

Conditions for time dependency, expressed for 2-phase controllers, do not exist in this case. According to (4) and (5) one-values of Req_{true} and Req_{false} grow up in correct moment, i.e. when *ReqCJ* arose. Therefore for Req_{true} and Req_{false} creation only one de-multiplexer is needed, managed by *CJ* signal.

The final description of automat into conditional jump branch is presented on Figure 6.

Stages into the rest linear sections of the pipeline are controlled by automats with logical structure shown on Figure 5.

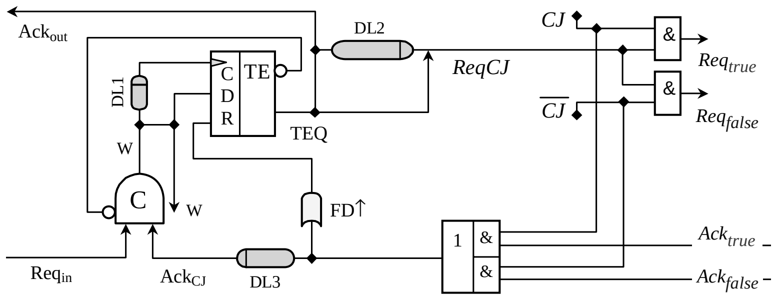


Fig. 6. Logic scheme of 4-phase pipeline controller into conditional jump stage

Conclusions and future work

The possibility for design of computational process with various algorithmic structures by the methods of micro-pipeline organization allows significant increasing of the performance. This is a result of the possibility for hardware implementation on one hand and on the other, because of the pipeline organization itself, which is basic method for entering parallelism into computations.

Although the presented in this paper aspects of the problem for computational process control in alternative conditions received decision, the problem of micro-pipeline implementation of common algorithmic structures was not completely solve. It can be assumed as resolved with the presence of decisions for the other aspects, defined in the beginning. These four aspects are strongly related and do not have practical independency.

References

1. Chammika Mannakkara, Tomohiro Yoneda, *Asynchronous Pipeline Controller Based on Early Acknowledgement Protocol*, National Institute of Informatics, Technical Riport, ISSN 1346-5597, Sept. 2009, Tokyo, Japan.

2. Feng Shi, Yiorgos Makris, Steven M. Nowick, Montek Singh, *Test Generation for Ultra-High-Speed Asynchronous Pipelines*, Int. Test Conference, pages 39.1–39.10, Nov 2005.

3. Singh M., Nowick S.M., *MOUSETRAP: Designing High-Speed Asynchronous Digital Pipelines*, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Volume 15, Issue 6, June 2007. http://www.cs.columbia.edu/~nowick/columbia-cisl-seminar-mousetrap-pt2.pdf .

4. Ozdag R.O., Singh M., Beerel P.A., Nowick S.M., *High-Speed Non-Linear Asynchronous Pipelines*, Design, Automation and test in Europe Conference and Exhibition, 04-08 III 2002, Paris, France, Proceedings, ISBN: 0-7695-1471-5, pp. 1000-1007.

5. Fawaz K., Arslan T., Lindsay I., *Conditional Acknowledge Synchronization in Asynchronous Interconnect Switch Design*, 2009 NASA/ESA Conference on Adaptive Hardware and Systems, Issue Date: July 2009, pp. 126-131.

6. Fu-Chiung Cheng, Shu-Ming Chang, Chi-Huam Shieh, *Detection and Generation of Self-Timed Pipelines from High Level Specifications*, 20th International Conference on VLSI Design (VLSID'07), January 2007, pp. 413-418.

7. Beerel P.A., Ozdag R.O., Ferretti M., *A Designer's Guide to Asynchronous VLSI*, ISBN 978-0-521-87244-7, Cambridge University Press, 2010.

8. Tyanev D.S., Josiffov V., Kolev S.I., *Operational structures without controlling automata*, International Workshop on Network and GRID Infrastructures, 27-28 Sept 2007, Bulgarian Academy of Sciences, Sofia, Bulgaria.

9. Tyanev D.S., Kolev S.I., Yanev D.V., *Micro-pipeline Section For Condition-Controlled Loop*, International Conference on Computer Systems and Technologies - *CompSysTech'09*, 18-19 June 2009, Ruse, Bulgaria, pp. I.4 1-5.

10. Tyanev D.S., Yanev D.V., Kolev S.I., *Method for realization of self-controlling loop apparatus structures*, Fifth International Scientific Conference *Computer Science'2009*, 5-6 November 2009, Sofia, Bulgaria.

11. Tyanev D.S., Popova S.I., *Asynchronous micro-pipeline with multi-stage sections*, ICEST'2010, 23-26 June 2010, Ohrid, Macedonia.

12. Tyanev D.S., Kolev S.I., Yanev D.V., *Race Condition free Asynchronous Micro-Pipeline Units*, International Conference on Computer Systems and Technologies - *CompSysTech'10*, 17-18 June 2010, Sofia, Bulgaria.

13. Kolev S.I., Tyanev D.S., *Early set to zero micropipeline*, International Conference on Computer Systems and Technologies - *CompSysTech'10*, 17-18 June 2010, Sofia, Bulgaria.

14. Sutherland, Ivan E., *Micropipelines*, http://www.jdl.ac.cn/turing/pdf/p720-sutherland.pdf .

15. Kinniment D.J., *Synchronization and Arbitration in Digital Systems*, John Wiley & Sons, ISBN 978-0470-51082-7, 2007.

Надійшла до редакції 01.03.2011