This paper considers problems related to hardware implementation of computational process with conditional jumps. Hardware refers to asynchronous pipeline organization at microoperational level. Exploration is dedicated to one of the tasks presented in (Tyanev, 2009) concerning to micropipeline controller design to control micropipeline stage into joint dot of branch algorithm. Joint dot is the point at which few preceding branches are combined. It appears inevitably into conditional jump structures and this is the reason for the actuality of its problem. Analysis of this new task is presented and request arbitration functioning principles are formulated for the incoming to joint dot requests. The arbiter is responsible for the fair choice on which depends steady performance of separate pipeline branches. Paper also describes pipeline controller synthesis and analysis of its operation in two variants: about 2-phase and 4-phase data transfer protocol. The synthesized asynchronous arbiter scheme is invariant to the type of pipeline protocol.

**Keywords:** Micropipeline, conditional jump, joint dot, pipeline controller, 2-phase and 4-phase protocol.

**Problem definition**

The paper considers exemplary algorithm which implementation is pipeline organized and has a structure as it shown on Figure 1. It is also assumed that this algorithm is described in details and its executable blocks are implemented by methods, presented in (Tyanev, Josifov, and Kolev, 2007), (Tyanev, Kolev, and Josifov, 2007), (Tyanev, Kolev, and Yanev, 2009), (Tyanev, Yanev, and Kolev, 2009) and (Tyanev, Kolev, and Yanev, 2010). Each executable block from the scheme represents single one- or multi-cycle micropipeline stage according to definitions expressed in (Tyanev, 2009), (Tyanev and Popova, 2010) and (Kolev and Tyanev, 2010).

As it is seen, presented algorithm can be described as branched. The condition for transition CJ (conditional jump) creates into the exemplary algorithm following possible branches for the computational process:

1. Begin; 1; 2; (CJ=true); 3; 6; End .
2. Begin; 1; 2; (CJ=false); 3; 4; 5; 6 ; End ;

where 1, 2, 3, … show the number of levels crossed by the execution.

The problems, caused by attempts for micropipeline organization of more common algorithmic structures, such as branched from Figure 1, are analyzed and considered in other our researches: (Tyanev, 2009), (Tyanev and Bozhikova, 2011).

Current paper treats one of stated into (Tyanev, 2009) new problems, related to micropipeline implementation of algorithms with general structure. Such micropipelines are described as non-linear. Here we present the decision about so-called joint dot. By “joint dot” we understand the point where algorithmic transitions from few previous branches are combined. The input of the micropipeline stage from level 6 in Figure 1 is
example for joint dot, because it combines outputs of two branches from the exemplary algorithm.

**FIGURE 1. EXEMPLARY ALGORITHM STRUCTURE**

Main problem of the joint dot, in view of receiving micropipeline stage, is that requests (Request) attending obtained in particular branches intermediate results are few from one side, and are concurrent in time from other side. The worst possible case for the pipeline controller, which receives these requests, is the case of their appearance at the same time. Each request presents demand for use of micropipeline stage at this level. Pay attention that requests which reached the joint dot compulsory are from different tasks, started previously into pipeline.

It is becoming clear that the synthesis of joint dot pipeline controller has following types of links:

1. Controller must accept all requests from previous parallel placed pipeline branches, i.e. it must have many inputs;
2. With respect to requests, pipeline controller has to distribute in opposite direction the actual value of Acknowledgment signals. These signals in opposite direction must reach the pipeline controllers in each of the previous branches. But the real acknowledgement must be received only from the controller in the branch, which request was accepted in accordance with 1. Acknowledgement signals have to be so much as request inputs;
3. From micropipeline stage side, which is the next stage at joint dot, pipeline controller has to use well-known pair signals Req_{out} and Ack_{in} to implement transfer protocol.

Above considerations present the hardware aspect of the problem. But it also has fully algorithmic aspect. The latter characterizes the choice of one particular request. This choice was described in (Tyanev, 2009) as arbitrator task, taking into consideration inevitability of request argument for common resource. Request arbitration task is principally known and there are different decisions, most of machinery - (Kinniment, 2007), (Patterson and Hennessy, 2005), (Hennessy and Patterson, 2003), (P6 Family of Processors Hardware Developer’s Manual), (Intel 64 and IA-32 Architectures Software Developer’s Manual); but here, in conditions of micropipeline control, we assume this task as a new one, because there will be defined special considerations for it. Presented information for the pipeline controller at joint point could be illustrated by Figure 2.
Each pair signals $\left(R_{j_{in}}, A_{j_{out}}\right), j = 1, k$ of pipeline controller PC is connected to preceding joint dot corresponding branch. At appropriate moment the controller creates signal Write and writes initial data into pipeline register of the stage at joint dot, which starts the computations into it.

**Arbitration principles formulation**

The fundamental moment into the problem under consideration is the choice among few request, received at the same time so it is necessary to define clear principles and criteria to search technical decision.

At first place, we will note that requests $R_{j_{in}}, j = 1, k$ at the input of the pipeline controller are absolutely equal, i.e. with same priority. Our understanding about this situation cannot be another, because between different tasks into pipeline there is no relation at all. Of course, this fact complicates the task for choice implementation.

At second place, it is not possible to reject the right of unconditional attendance, if the fixed request is one. At such moment this request gains in time the right to forward its task computations further into pipeline.

At third place, there must be some justice in case when there are few requests and one of them again is that was previously chosen. We could assume the exception of this request from the current choice. In this manner we would give a chance to some of the remaining tasks to move into pipeline. Of course, this temporal prohibition must be cancelled in case the repeated request is unique. If this request gains the current competition (choice), it will have the right to move forward its task into pipeline.

At fourth place, on the occasion of the presented limitation, there is a question about how back in time could be one request attended if at arbitration every time there are all $(k)$ requests available? Let’s remind that in fact it is all about the choice of task to continue its execution into micropipeline state at joint dot. This means that each request, which is not chosen, has to be supported in state of expectation from its pipeline controller. In this way it has to be clear that the task, which execution had pass given branch, is temporary stopped and will wait its turn at this point. All the others not chosen tasks will be in the same state. They will wait for stage release at joint dot, when its controller will make serial choice. Meanwhile, it is possible that previously chosen and attended request to join to the waiting service requests.

Now we can say that if at joint dot there are such extreme situations, as fair choosing strategy we define the following: requests will be chosen consecutively in round with constructive set order. The order may correspond to natural numbering of the branches $(1, 2, \ldots, k)$, which will make conditionally the constructor. This strategy guarantee
attendance of request from random branch after not more than (k-1) previously serviced requests from the other branches.

We will illustrate upper statement as follows: let into RG-Req at certain moment there are conditionally registered requests from corresponding branches, as it is shown on Figure 3. The diagram shows situation when stages into branches from 1 to (k-5), as well as into branches (k-3) and (k-2), are busy at the current moment with their computations and are not passing requests to the pipeline controller at joint dot. This is the reason why corresponding flip-flops into RG-Req register are in state “zero”. At the same time, the end stages into branches (k-4), (k-1) and (k) have finished their computations and are registered the requests to the joint dot controller.

**Figure 3. Choosing cycle for registered stages**

In this background, request $R_{k-1}$ would be the next to choose if the currently chosen is $R_{k-4}$. If the consecutive choice is request $R_{k-4}$, as it is shown on the figure, flip-flop (k-4) of the chosen request must be immediately set to zero. This is necessary on one hand for implementation of defined choosing strategy and for this branch’s next request registration from other.

While the stage at joint dot executes newly accepted request $R_{k-4}$, it is possible to be received and registered requests $R_{k-3}$ and $R_{k-2}$. So at next choice instead of $R_{k-1}$ will be chosen $R_{k-3}$, which in fact is jump the queue. Even more, the request $R_{k-1}$ is anticipated by requests $R_{k-3}$ and $R_{k-2}$. So, under the strength of consecutive choosing order, the request $R_{k-1}$ must wait more than was expected. Despite of this, the expectance will not be more than mentioned (k-1) previous attendances.

If the requests cannot make described jump of queue, the arbiter has to support separate register RG-F, which will fix and store all new requests. RG-F will store and update its condition till the moment in which all flip-flops into RG-Req will be in state “zero”, i.e. when the cycle of going round will finish. At this moment RG-Req has to accept requests from RG-F and to start new cycle. From its part, after reset RG-F again will start registration of new requests.

It is easy to consider, that in both variants of consecutive attendance the equal status of requests allows not more than one service at one cycle. That is why we will search for the simplest technical solution.

**Structural design of joint dot controller**

Micropipeline stage at joint dot must:

1. Accept data $D_j$ from chosen $j$-branch. In order to complete this, data stage register has to be preceded by multiplexer, denoted as MUX-D;
2. Accept chosen request $R_{j_in}$ as a signal $Req_{in}$ to pipeline controller, which means presence of multiplexer for input requests, denoted as MUX-R;
3. Produce acknowledgement signal $\text{Ack}_{\text{data}}$, which will distribute as signal $A_{j \text{out}}$. The latter must be received only by controller of the chosen $j$-branch. In the direction of controllers which does not receive this acknowledge, the joint dot stage controller must keep permanently in time the value of the last sent acknowledgement. This means that acknowledgement values have to be kept into flip-flops. The access to the separate flip-flops will be provided by the arbiter through generated choice code. These flip-flops will create acknowledgement register $\text{RG-Ack}$.

Management of these logical elements depends on the choice, which means that the arbiter will be part of the joint dot logical structure.

**Pipeline controller with 2-phase protocol**

The purpose of pipeline controller at joint dot is to produce in the right moment signal $W$ for writing $D_j$ data, coming out of $\text{MUX-D}$, to pipeline stage register. Design of this controller is related to emerging logical structure for stage control at joint dot. Special attention will be paid to the event-driven nature of processes into the structure.

**Incoming requests synthesis**

Incoming requests $R_{j \text{in}}$, $j = 1, k$, when they are generated by 2-phase controllers, are potential signals with active both logical values (Sutherland, 1989). With this protocol operative is pipeline controller edge in the switching moment (front-end or back-end). Each new switching of the pipeline controller causes data write into pipeline register and in this way - start of stage computations. This is the reason why pipeline registers must be realized by $\text{DEDFF}$ flip-flops (Appleton, Morton, and Liebelt, 1997), (Sutherland, 1989). But in our case, when there are many requests to the pipeline controller, it is necessary choice to be made. Different and unpredictable levels of incoming requests definitely create additional problem.

On other hand, the appearance of acknowledgement signal for stage readiness at the joint dot has also event-driven nature. So we assume that for the choice decisive is not the moment of new request appearance, but the moment when the acknowledgement for stage readiness at the joint dot appears, because this is a consequence of its controller switching and is start for new choice. Pay attention that in 2-phase protocol the acknowledgement signal is also potential with active both logical values.

Presented considerations give proof of necessity for fixing in time of the new requests. These new requests have to be asynchronously registered into separate register $\text{RG-Req}$, which was commented into section II. In view of the above we can claim that $\text{RG-Req}$ will be implemented with asynchronous $\text{RS-Latch}$ flip-flops. In order to detect the switching moment of the request value, storing of the new switch into flip-flop of $\text{RG-Req}$ will be done by two edge-detection schemes: $\text{FD}^\uparrow$ for front-end and $\text{FD}^\downarrow$ for back-end.

$$ S_j = (\text{FD}_{\text{Req}j}^\uparrow) \cup (\text{FD}_{\text{Req}j}^\downarrow) . $$  \hspace{1cm} (1)

Logic for reset of the flip-flops in this register will be described later. At this moment it may be told that initially this will be done by compulsory $\text{Reset}$ signal. After registration, each new request exists at the output of its own flip-flop into $\text{RG-Req}$ as 1. The zero value in flip-flop form this register must be understood as lack of new request. In other words, the ones will be assumed as unserviced requests from which to choose, and the
zeros - as missing requests. As a conclusion, RG-Req appears as a peculiar transformer of request level - from random they will be temporary stored only as ones.

Fixed requests will be lead to arbiter which will generate the number of the chosen request. The number will be used as code for control of MUX-R multiplexer, from which will go out the chosen request $R_j$; also for control code of MUX-D, through which will pass the data from the chosen branch $D_j$, as well as control code for acknowledgement $A_j$ write to the chosen branch in register RG-Ack. The logical structure of stage control at joint dot is shown on Figure 4.

**FIGURE 4. STRUCTURE AT JOIN DOT**

If we assume, that at certain moment stage pipeline controller at joint dot will be switched, i.e. the stage will receive new data and will start the computations, and it means that its controller has received the acknowledgement $Ack_{in}$ (Figure 4). In view of the fact, that chosen by the arbiter request $R_j$, which must be transformed into $Req_{in}$, will have random logical value, we must convert this request in the way to realize correct the forthcoming controller switching. For this purpose we create the following truth-table:
TABLE 1. LOGICAL VALUE OF REQUEST \( \text{REQ}_n \)

<table>
<thead>
<tr>
<th>( \text{Ack}_n ) arrives with new value</th>
<th>C-element is in state</th>
<th>Chosen request ( R_j ) has value</th>
<th>C-element must switch to state</th>
<th>For this purpose ( \text{Req}_n ) must be</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \sqrt{1} )</td>
<td>0</td>
<td>0</td>
<td>( \sqrt{1} )</td>
<td>1 (( R_j ))</td>
</tr>
<tr>
<td>( \sqrt{1} )</td>
<td>0</td>
<td>1</td>
<td>( \sqrt{1} )</td>
<td>1 (( R_j ))</td>
</tr>
<tr>
<td>( \backslash0 )</td>
<td>1</td>
<td>0</td>
<td>( \backslash0 )</td>
<td>0 (( R_j ))</td>
</tr>
<tr>
<td>( \backslash0 )</td>
<td>1</td>
<td>1</td>
<td>( \backslash0 )</td>
<td>0 (( \overline{R}_j ))</td>
</tr>
</tbody>
</table>

Logic of filling this table is following (see the first row):

- If the acknowledgement \( \text{Ack}_n \), which is received by joint dot form the next (lower) stage, is one (appears front-end switching);
- And if the controller is in zero state;
- It can be switched only to one-state.

To implement controller switching to one, the \( \text{Req}_n \) signal value must be one. If in this situation the chosen \( R_j \) is zero, then \( \text{Req}_n = \overline{R}_j \). Likewise, the remaining rows in the truth-table are filled.

For the table above we must explain that the new value of \( \text{Ack}_n \) signal cannot be combined with the same value for the C-element state (first and second columns), because they create set, referring one and the same controller.

From truth-table can be seen that in both cases, when \( \text{Ack}_n = R_j \), the necessary value is \( \text{Req}_n = R_j \), and when \( \text{Ack}_n \neq R_j \), the inverse value \( \text{Req}_n = \overline{R}_j \) is needed. Then for the signal \( \text{Req}_n \) follows next logical equation:

\[
\text{Re}_{q_n} = (R_j) \cap \overline{\text{Ack}_n} \oplus \overline{R_j} \cup (\overline{R}_j) \cap (\text{Ack}_n \oplus R_j)
\]  (2)

The equation (2) defines the logical scheme of the convertor \( \text{Conv} \) (Figure 4).

**Outgoing acknowledgement synthesis**

After its switching, the stage pipeline controller at joint dot returns acknowledgement signal \( \text{Ack}_{out} \). However this signal has to reach only the controller in \( j \)-branch, which request was accepted. It was already explained, that the logical value of this signal is not actual, because it is most likely that it doesn’t correspond to the expected for \( \text{Aj}_{out} \). The reason for this is 2-phase protocol from one side and set of parallel links, connected to the stage at joint dot, from other hand. As we mention also the unknown number of preceding switching, as well as their order, then the probability for logical value to corresponds to the expected, is insignificant. The logic value of acknowledgement \( \text{Aj}_{out} \), which must be returned back to \( j \)-controller, depends from this controllers’ state. If it is one (\( G_j = 1 \)), the request value to be transmitted is also one, i.e. \( R_j = 1 \) and vice versa. Because each switching of the Mueller C-element is opposite, the new value of incoming acknowledgement is opposite to the state, and from there to the request. Then
But this value must be passed in time in a moment, determined by the switching moment of $\text{Ack}_{\text{out}}$ signal. Passing of the necessary acknowledgement value in the corresponding moment can be achieved only with the corresponding flip-flop (look Figure 4) from register RG-Ack. The moment is indicated by switching edges for C-element at joint dot, through two edge-detectors $\text{FD}_\uparrow$ and $\text{FD}_\downarrow$, which disjunction realizes the load in this D flip-flop. The inverse output of this flip-flop implements decision (3). Because the acknowledgement value (3) is passed to D-inputs of each flip-flop in RG-Ack register, as a signal enabling load to the current flip-flop $j$ is used generated from the arbiter code (Bus $\text{Ne}$). Logical scheme of one bit from RG-Ack is shown at Figure 5.

**Arbitration scheme synthesis**

Typical arbitration is made on the base of priority by one-cycle logical scheme, known as daisy-chain (Kinniment, 2007), (Yun-Lung Lee, Jer Min Jou, and Yen-Yu Chen, 2009), (Shin, Mooney, and Riley, 2002), (Patterson and Hennessy, 2005) or (Hennessy and Patterson, 2003). In these cases arbitration is made with external initiative, executing the choice in the arbitrating scheme. However, in our case priority introduction is not reasonable, so it was decided requests to be chosen consecutively in round order. Each new choosing procedure must start from the next position ($j+1$). According to the shown of Figure 3 example, after $R_{k-4}$ request reception next choosing procedure must start from position ($k-3$). In contrast to solution in (Procopov and Tyanev, 2009), where this sequence is achieved by special blocking code, here we apply inactive logical choosing scheme (Tyanev, 2008) and two registers for requests, which were commented in section II. The choosing scheme is inactive because it doesn’t use external signals. The choice made by this scheme must be supported continuous in time till the next change into incoming requests set. As example, on Figure 6 we present synthesized for this purpose logical scheme of asynchronous arbiter with 4 inputs.

Arbiter with another length can be easily received based upon presented here scheme by adding or removing of corresponding groups of logical elements.

The scheme of this arbiter is original and entirely asynchronous. As it is proven in (Procopov and Tyanev, 2009), whole procedure between two choices elapses in time, because is multi-cycle. Particular stages for proposed scheme will be described later.
As it is seen, the arbiter contains two registers from asynchronous RS-Latch flip-flops - RG-Req and RG-F. New incoming requests at inputs R0, R1, R2, and R3 are fixed in register RG-F by single impulses, generated by edge-detecting schemes $FD_{\uparrow \downarrow}$.

Inactive scheme, which accomplish the choice of current request, is connected to the outputs of RG-Req. The scheme chooses one request among registered, consecutively for request from 0 to 3, as we wanted in the beginning. Because of the pair registers with distinguishable functions jump of queue is not possible for new request. Described

**Figure 6. Logical circuit of 4-input arbiter**
discipline is possible in condition when there is immediate reset of the trigger, supported the chosen request till the current moment, after the acknowledgement. That is why the reset is realized by signal Ack-FD↑↓, which is sent back in response of accepted request right after pipeline controller switching into the new state. This process is shown on Figure 7.

Figure 7 presents both possible cases for 2-phase protocol - switching of the acknowledgement Ack_{out} to zero and its switching to one. The delay of the new request (j+1) for example is equal to the sum from delay of resetting trigger and delay of choosing scheme.

**Figure 7. Reset of chosen request**

![Diagram showing reset of chosen request](attachment:image.png)

In the time of described actions, the micropipeline stage at joint dot is making its new computations. So after the signal Ack-FD↑↓, which removes from the list accepted request, the choosing scheme chooses new request and readjusts multiplexers MUX-D, MUX-R and demultiplexer DMUX by the bus Bus_{N}. From its part, the new chosen request R_{j} reaches converter Conv and then reaches pipeline controller, where it will wait for the next acknowledgement Ack_{in} for the joint dot. It is clear, that one choosing procedure elapses between two acknowledgements Ack-FD↑↓.

**Figure 8. New portion requests write**

![Diagram showing new portion requests write](attachment:image.png)

Regardless of how many are the ones into RG-Req register, at the four scheme outputs there will be only one 1. It corresponds to the chosen request R_{j}, j=0,1,2,3. If the RG-Req content is zero, on the scheme outputs there are four zeros. When this situation occurs, it means that the consecutive portion of requests is attended and the next has to be accepted. In the moment when there are four zeros at the scheme outputs, the decoder
EQ1 creates outgoing zero. As a result of this switching, the edge-detector placed after the scheme generates single impulse FD-EQ↓, which makes new write into RG-Req after additional delay DL. In this way, fixed into RG-F requests are transferred to RG-Req and over them start new arbitrating cycle. The delay DL of writing impulse Write is necessary in order to avoid eventual contradiction between microoperations reset and write, which follows directly one after another over the same flip-flop. This part of the process is illustrated by the time diagram from Figure 8.

This time diagram also shows both possible for 2-phase protocol situations. The cycle of attending 4 requests is in the interval begin-end. It can be seen, that the delay DL of the impulse Write must be greater than the period of edge-detector impulse FD-EQ↓. This condition is required for reliable write.

4-bits output bus form choosing scheme BUS № is used into the structure from Figure 4 for MUX-R and MUX-D multiplexers’ control, as well as for control of E-inputs of flip-flops into RG-Ack.

**Arbiter special situations**

At the beginning will be described the microoperation reset. Each new request choice, which is made by the arbiter, is assumed as completed with receiving of the acknowledgement Acknat (see Figure 4). With appearance of this acknowledgement only the triggers of serviced request are reset into both registers RG-Req and RG-F. This is achieved by the single impulses of signal Ack-FD↑↓ (Figure 6). The register RG-Req, as well as the register RG-F, is embraced on R inputs of its triggers by feedback of the bus BUS №. On this feedback, next to AND-schemes, before resetting inputs of the flip-flops, the one of the new chosen request appears. But before this, the one passes through delay DL. Such delays are included in the routes of all four requests. These delays served its purpose in the moment of trigger reset for the accepted request in the following manner - with the reset of corresponding trigger into RG-Req, starts the choosing scheme, connected to its output, realizes new choice and the new one reaches through the feedback to AND-scheme, placed before resetting input of its trigger, where at the same time is coming the reset impulse. Combination in the time of these two 1 at the R-input of this trigger is illegal, because request supported by the flip-flop still is not accepted for servicing. This is the reason why this trigger’s state has to be stored, which is possible only if the delay DL of the new chosen request is greater than the feedback’ delay and the period of reset impulse Ack-FD↑↓. In other words, the new request and reset impulse must pass each other in the time in order to avoid the switching of AND-scheme of flip-flop, which must not be switched. Described situation can be observed in the time diagram on Figure 8.

The flip-flop reset for accepted request in RG-F is made in different conditions. It can be seen, that it is possible only if in the moment of impulse Ack-FD↑↓, there is no other request for fixing from the same source. The appearance of new request registration impulse from source, which former request was newly accepted for servicing, is impossible. This is because such request Rfut (Figure 6) could be received only as a response of acknowledgement Acknat, which has to be send through DMUX as acknowledgement Ajnat for which the micropipeline stage into corresponding branch doesn’t have time. In other time this third input of AND-scheme is always allowed with one.

Next special situation to describe is related with the moment of writing the new portion requests from RG-F to RG-Req. In particular, we consider situation when there is no such requests registered into RG-F till moment. It means that after servicing the last request from preceding cycle and after resetting of corresponding flip-flops, the register RG-F occurs empty. So, the following these events impulse Write will write zeros into RG-Req, i.e. will not change its content, which at this moment is zero as well. For this situation, it
signifies that the end micropipeline stages in all branches, preceding joint dot, are still busy with their current computations and does not pass requests. At the same time, all previous requests are already serviced.

Because the signal Write is already used in this situation, besides without success, and cannot be generated again, the arbiter remains in waiting state. Processes can be started again only by initiate of new incoming requests, influencing directly the choosing scheme. The latter means that there must be direct request write into RG-Req. The direct write is provided by the common feedback of input requests $R_i$ (Figure 6) with two registers at S-input of the triggers. The write of the first received request is asynchronous and is provided with the permission, supported by EQ2. With the presence of request into RG-Req the consecutive arbitration is completed. The chosen request reaches pipeline controller into joint dot stage, where arrived a long time ago acknowledgement is waiting for it. Then the processes are repeated. Described special situation is illustrated by the time diagram at Figure 9.

**Figure 9. Write with request lack**

At Figure 9 is shown situation after back-end of Ack$_{out}$ signal. Likewise is the situation with its front-end. The new arbitration cycle starts in the moment, denoted as “begin”, but into described situation it will not wait for the moment “end” if there won’t be at least one request to appear and to be registered. The state without requests can be defined as initial. In the beginning it will be achieved by forced reset with Reset signal, which is not shown on Figure 6. On the time diagram, at the last x-axis is presented time interval, when receiving of new requests is possible, as well as development of the process after first new request appearance.
Pipeline controller with 4-phase protocol

Here we will present the alternative possible solution - pipeline controller with 4-phase protocol. The chosen in (Tyanev and Popova, 2010), (Kolev and Tyanev, 2010) pipeline controller and its protocol are shown on Figure 10 and Figure 11 respectively.

**Figure 10. Pipeline controller with pre-emptive reset**

The controller contains Mueller C-element which generates signal Write for data write into pipeline register and dynamically controlled trigger TE, realizing the dialog with neighbor pipeline controllers. At the time diagram are presented two situations of the protocol. In the first one, the controller has received the request Req\(_{in}\), but there is still acknowledgement Ack\(_{in}\) missing. The second situation is opposite - there is acknowledgement Ack\(_{in}\) available and waits for Req\(_{in}\) arriving. In both cases, after C-element detects two ones at the same time in these input signals, it generates transfer write signal W. After appearance of this signal and with its distribution to the TE trigger, into the scheme passes avalanche-like process and the controller is set into initial state. The signals Ack\(_{out}\) and Req\(_{out}\) are sent to the neighbor controllers.
The most special feature about this transfer protocol is that true values of signal Request, as well as of signal Acknowledgement, appear only as constant 1. Logical zero is assumed as lack of request or acknowledgement. Such situation creates some reliefs for presented in this paper problem. About micro-pipelines with 4-phase protocol, as well as for protocols with pre-emptive acknowledgement, for transfer control to the stage at joint dot we propose logical structure from Figure 12.

Consider the current process of the stage at joint dot. Let the arbiter choose request Rj. As a result of this choice, multiplexer MUX-R and demultiplexer DMUX are switched. So the pipeline controller of the chosen branch is connected to the stage controller at joint dot and $R_j \equiv Req_{in}$ and $A_{jout} \equiv Ack_{out}$.

![Figure 12. Structure of Joint Point Controller](image)

When into stage pipeline controller at joint dot is achieved its switching condition $Req_{in} = Ack_{in} = 1$, it will be switched and will create acknowledgement $A_{jout} = 1$, which as acknowledgement $A_{jout} = 1$ will be send back to the j-controller through DMUX into corresponding branch. According to the protocol we consider, this controller will immediately reset the request $R_j$, as it can be seen on Figure 11. In difference with 2-phase protocol case, only combinatorial DMUX is enough for $A_{jout}$ distribution into opposite direction. This scheme supports the actual acknowledgement value only on the chosen output. There are zeros going out of the remaining outputs which are taken as a lack of acknowledgement from the corresponding controllers. So these controllers support only their own request and the result obtained by their stage.

As a result of received in this way $R_j$, into RG-F of the stage at joint dot there is data $D_j$ written and is started consecutive computation. With the beginning of these computations, the arbiter must reset this request and to choose the next one.

Taking into consideration already presented conception for the arbitration itself, we can say, that designed logical structure (Figure 6) does not depend on the type of protocol
used for transfer between micropipeline stages. The only change which has to be done in this scheme is to place on request inputs of edge-detectors, working on front-end. This change is not a reason for new description of the scheme at Figure 6.

Conclusion

Presented in this paper decisions of the problem for synthesis of pipeline controller controlling stage at joint dot in micropipeline structure, supplements the set of solutions made for the general problem, described in (Tyanev, 2009) for development of methods, allowing micropipeline implementation of random algorithmic structures. The four new problems, defined there, have received its decisions in this paper, as well as into (Tyanev, 2009) and (Tyanev and Bozhikova, 2011). These solutions are described for 2-phase and for 4-phase pipeline protocols. Created logical structures can have many other variants and special features, which will be considered in the future.

References


