ASYNCHRONOUS MICRO-PIPELINE WITH MULTI-STAGE SECTIONS

Dimitar S. Tyanev[1], Stefka I. Popova[2]

Abstract: **The interface of multi-stage micro-pipeline sections building continuous micro-pipelines is defined and analyzed. As the multi-stage micro-pipeline sections have own memory, such micro-pipelines don't need additional registers. In these conditions there is pipeline asynchronous protocol and implementing control unit synthesized. The protocol's operation is shown in cases, arising from combined work of neighbor multi-stage micro-pipeline sections. Possible problems of combining one- and multi-stage sections are indicated.**

Keywords: **Computational devices, Micro-pipeline, Race conditions, Synchronization**.

I. INTRODUCTION

Micro-pipelines contain consecutively connected micro-pipeline sections which structure is made of register and logic (for example [6÷18] or another). The register supports the data and the logic implements the necessary computations, but it is not required. If the delays describing the particular sections are relatively the same, there is common control and the micro-pipeline is defined as synchronous. If the delays are significantly different, the control is distributed and the micro-pipeline is determined as asynchronous. After every registration impulse new data enters and is processed at certain section in the both types of micro-pipelines. So after each impulse the intermediate results are moving from section to section. In these terms the micro-pipeline sections in such kind of pipelines are defined as one-stage. The stage period at particular sections is set by the switching time of the logic.

Data shifting from section to section in asynchronous micro-pipelines of mentioned type is implemented after the hand-shake principle, in 2-phase or 4-phase protocol. The protocol is realized by control block, containing some version of well-known Mueller C-element [5÷18]. The nature of control is asynchronous because the shifting of current results to the next section is possible only if the last is not busy. This is the main reason such type of micro-pipelines to be defined as asynchronous.

Micro-pipeline sections (MPS) with internal feedback are presented in [1÷4]. These sections implement iterative computations and are designed as synchronous devices. They work as synchronous because of local clock. Such micro-pipeline sections can be determined as multi-stage on account of the internal (local) clock. The delays generated from these sections are significantly different amongst themselves, as well as compared to the delays from one-stage sections, so as devices they can be included only in asynchronous micro-pipelines. As the multi-stage micro-pipeline sections have their own memory, the micro-pipelines with such sections don't need additional pipeline registers. This paper presents the interface of this kind of multi-stage micro-pipeline sections and the control possibility with serial inclusion.

[1]Dimitar S. Tyanev is with the Faculty of Computer Sciences and Technologies, Technical University of Varna, Bulgaria.
e-mail: dstyanev@yahoo.com

[2]Stefka I. Popova is with the Faculty of Computer Sciences and Technologies, Technical University of Varna, Bulgaria.
e-mail: s_ivanova@abv.bg

II. MULTI-STAGE MICRO-PIPELINE SECTION

Micro-pipeline sections with internal feedback can be presented by the following general structure:

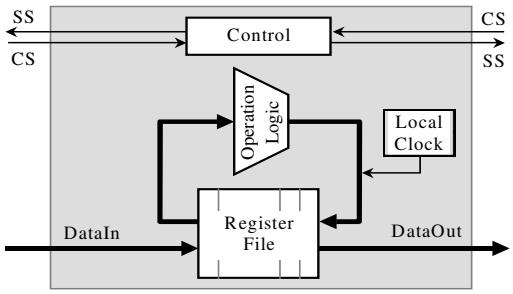

Figure 1 Structure of multi-stage micro-pipeline section

The structure contains three basic elements – register file (*Register File*), which consists of one or more registers (pipeline fixers) and set of combinatorial logic (*Operation Logic*) implementing necessary computations. The main characteristic of this structure is the internal feedback. The third element (*Control*) is integral part of such sections and realizes their internal control. Some of the tasks of the internal control will be discussed in this paper. The most important function of the internal control is to carry out the communication between sections and to process and generate the respective signals (*Status Signals, Control Signals*).

Multi-stage micro-pipeline sections are intended to implement various types of cyclic algorithmic structures. The iterative computations carried by such sections lead to long detention of the main computational process, which allows them to be defined as very asynchronous regarding to it.

Micro-pipeline sections of presented type can be stable in one of the next three states:

1. "Free" state. It means that the result of computations in the current section *k* is sent through the output bus *DataOut* to the next section *k+1* and the last confirm the reception. In this state the logic connections in the structure are determined so the section is ready for the next start. In these terms, the state "Free" is the information necessary to section *k-1*, because there is no sense to be started if the next is not free;

2. "Busy" state. Current section *k* is in this state during the implemented cyclic computations. At this time the input data bus *DataIn* is switched off and the data on it does not have any impact on its structure. The data on its output bus *DataOut* is not valid so it shouldn't be accepted and used by the next section;

3. "Ready" state. It is alternative to the previous state. It occurs in the current section *k* when its computations finish and the true value of the result are set on the output bus *DataOut*. In this state the section supports the obtained result on the output bus so it is still only there.

Note: All micro-pipeline sections in certain micro-pipeline should be forced to "Free" state after the power switching, as well as in other situations that require this state. The last is defined as initial or last state for the particular section and for the pipeline in general.

At the time of the pipeline operation the order of the states in each section is as follows:

... "Free", "Busy", "Ready", "Free", "Busy", "Ready", ...

States in which every section of the micro-pipeline could be are declared by the following signals (signals of SS type):

1. Signal F_k (Free). It is produced after switching of the section in "Free" state. This signal is conditionally directed to the back, i.e. to the previous section *k-1*;

2. Signal B_k (Busy). It is produced after switching of the section in "Busy" state. The signal is directed to the back as well;

3. Signal R_k (Ready). This signal is produced after switching of the section in "Ready" state. It is conditionally directed to the front, i.e. to the next section *k+1*.

III. CONSECUTIVE INCLUSION OF MULTI-STAGE MICRO-PIPELINE SECTIONS

Multi-stage micro-pipeline sections are included in exact sequence according on the current algorithm. In order to the pipeline organization there is certain control required, depending on the signals SS and CS. The control of the connection between each pair neighbor sections is assigned to control automation (CA) which must recognize the states of the both sections and to manage their dialogue. In other words, this automation has to synchronize the common work of two neighbor sections using signals of SS type and in response to produce the necessary CS-signals (Figure 1). In conformity with Section 1, the pipeline automation should be asynchronous. Figure 2 presents pipeline from the discussed type.

As it seen from the figure, the control automation CA generates two control signals:

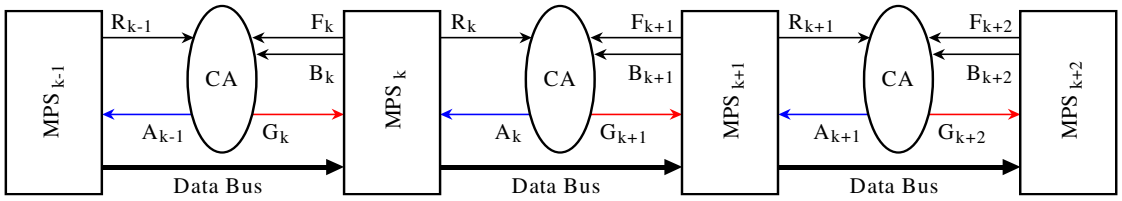

Figure 2 Micro-pipeline with multi-stage sections

1. Signal G_{k+1} (*Go*). With this signal the automation starts computations into the next micro-pipeline section, i.e. the signal is directed to the front. The emission of this signal must be possible only if the previous section is in "Ready" state and next – in "Free" state. This is situation in which the previous section finished its computations and supports the results on the output bus. At the same time the next section is free and waits for new data;

2. Signal A_k (*Acknowledgement*). With this signal the automation informs the previous section that transferred data is successfully received by the next one. The signal is conditionally directed to the back. The previous section must announce "Free" state in response. The logic of such state is presented as:

$$F_k = R_k \cap A_{k+1} \quad . \qquad (3.1)$$

The control automation's operation is showed by the graph at Figure 3. The graph shows that the automation has two states. The initial state is marked as *S0*. In this state the automation supports the signal A_k which is a reason for the "Free" state of section *k*. From this state automation is switched to state *S1* only when the two neighbor sections complete the required transition condition: $R_k \cap F_{k+1} = True$. Once automation is in *S1* state it produces the signal G_{k+1}. This signal appears to be initial for the next section *k+1*.

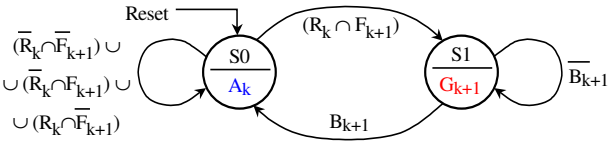

Figure 3 Transition graph in CA

After section *k+1* begin its operation, it passes to "Busy" state and forms signal B_{k+1}. This signal causes the switching of the synchronizing automation back to the initial state *S0*. From this state is produced signal A_k which informs the previous section about the successful transfer of its data to the next section. It is the end of the exchange session at this stage of the micro-pipeline.

The synthesis of the pipeline automation in terms of the transition graph from Figure 3 leads to the principal logic structure presented at Figure 4 in two variants:

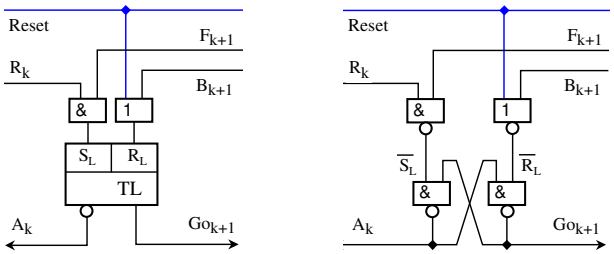

Figure 4 Principal structure of CA

The automation is implemented following Moore's structure by one asynchronous RS-Latch flip-flop. Internal states of the automation are coded as follows:

$$S0 = \overline{Q}\ , \quad S1 = Q\ . \tag{3.2}$$

So the right input implements signal *Go* and the inverse input – signal *Acknowledgement*.

Because of the different duration of the computations in two neighbor sections there are two possible situations for the control automation, for example:

1. Section *k+1* is free and waits for the end of section *k*'s computations;
2. Conversely, the section *k* is ready and waits for the end of computations in section *k+1*.

The time-diagram from Figure 5 shows these two cases of the pipeline automation switching. In the first situation (the left half) section *k+1* waits for the data from section *k*. The automation is in *S0* state, waiting for signal R_k.

In the second case (the right half) section *k+1* still works while the previous section *k* finish computations and is in "Ready" state at the same time, producing signal R_k. With this signal the automation is switched to *S1* state from the *S0* state.

The transitional process corresponding to automation's graph is presented twice – into the left and into the right side of the time-diagram at Figure 5 and shows the beginning, work, final and repeated start of the micro-pipeline section *k+1*. Analyzing this time-diagram can be concluded that the control automation implements 4-phase protocol.

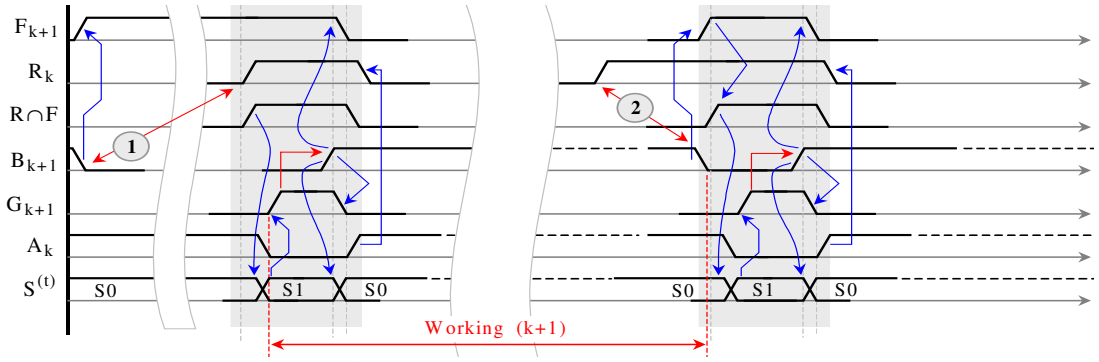

Figure 5 Switching of the synchronization automation

IV. SYNCHRONIZER

For starting the computational process in multi-stage micro-pipeline sections [1], [2], [3], [4] is required start impulse, conditionally called *Enable* which must be synchronous with the rising edge of the local clock impulses and to have duration up to one period. From discussion in Section 3 is clear that the parent of the start signal will be the asynchronous control automation which generates signal *Go*. This signal is asynchronous regarding to the local clock impulses. Thus, there is a task for the signal *Go* converting into signal *Enable*.

The task of converting asynchronous signal into synchronous is illustrated by the time-diagram at Figure 6:

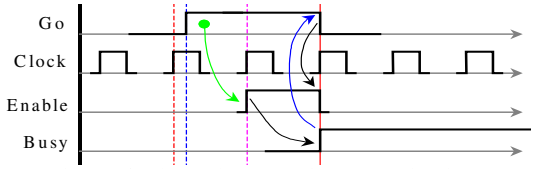

Figure 6 Time-diagram of synchronization

Can be seen that the signal *Go* appears asynchronously in the time of clock impulse from the Clock sequence. The start impulse *Enable* to MPS follows as response. With appearance of the *Busy* signal disappear the signals *Go*, which is function of the pipeline automation, and *Enable*, which should be function of the synchronization schema. Signal *Busy* characterizes the state of the micro-pipeline section, as it was described in Section 2.

The schema, which implements expressed logic, is presented at the following figure:

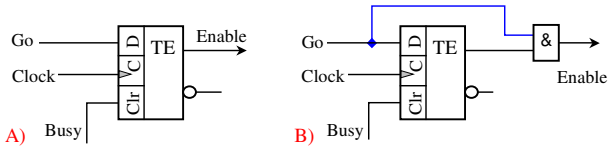

Figure 7 Principal structure of the synchronizer

There is dynamic D flip-flop with Edge structure used, which is basic approach for synchronization of asynchronous signals [15, 19, 20, 21, 22]. Fixing of the signal *Go*'s logic value is made with the rising edge of the clock impulse. If it is missed, as it shown at Figure 6, it could be done with the next impulse. For reliable fixing there must be restriction on the initial asynchronous value. This value should be kept in time during the following period:

$$t_{Go} \geq T + t_1\ , \tag{4.1}$$

where t_{Go} is the duration of the signal Go;

T is the period of signal Clock;

t_1 is the duration of single impulse into signal *Clock*.

If the signal *Go* has significantly bigger duration, in the name of the necessary duration of *Enable* signal is used forced cleaning of the flip-flop by the *Clr* (Clear) input. Notice that this input is with high priority and if the signal *Busy* is connected to it (*Busy* is active through all computational cycle), the forced keeping of the flip-flop in zero state is guarantee for reliability and makes impossible false values of the signal *Enable*.

There are two options for the synchronizations schema (look Figure 7). The variant A synchronizes by the rising and by the falling edge; the variant B synchronizes only by the rising edge. In the last variant, because the *Enable* signal is function of the input signal *Go*, it is the direct reason for its disappearance. The synchronizer should be assumed as part of the logic of every multi-stage micro-pipeline section.

V. CONCLUSION

Discussed in this paper type of micro-pipeline sections and their serial inclusion in certain micro-pipeline is only one special case. As it was told in the beginning, there are different types of possible micro-pipeline sections. It is possible their serial inclusion in various combinations, for example: one-stage section followed by multi-stage section or multi-stage section, followed by one-stage. If we consider longer sequence, the possible combinations will be more. The only one known case, which was mentioned in the introduction, corresponds to sequence of one-stage micro-pipeline sections, where the control automation is based on Mueller C-element.

While one-stage sections are served only by two signals, usually called *Request* and *Acknowledgment*, for the multi-stage sections is defined a few possible signals. However, knowing the structure of the multi-stage sections we can claim that there are preconditions for other interpretations, as well as for possibilities for generalization. Admitting the mentioned combinations of serial arrangement, it will lead to necessity of different kinds of control automation. Of course, these possibilities are topic of future work.

REFERENCES

[1]. Тянев, Д., Колев, С., Янев, Д., Метод за реализация на апаратни самоуправляващи се циклически структури - част II, "Компютърни науки и технологии", ТУ-Варна, ISSN 1312-3335, година V, брой №2/2007, стр. 23-30.

[2]. D. Tyanev, S. Kolev, D. Yanev, Micro-pipeline Section For Condition-Controlled Loop, CompSysTech'09, 18-19 June 2009, Ruse, Bulgaria.

[3]. D. Tyanev, D. Yanev, S. Kolev, Method for realization of self-controlling loop apparatus structures, Fifth ISCCS'09, 5-6 November 2009, Sofia, Bulgaria.

[4]. Тянев, Д. С., Колев, С. И., Йосифов, В., Метод за реализация на апаратни самоуправляващи се циклически структури, ТУ-Варна, ЮС "45 години ТУ-Варна", 2007, ISSN 1311-896X, стр. 130-135.

[5]. Миллер, Реймонд Е., Теория переключательных схем, том 2 – последовательностные схемы и машины, Москва, Издательство "Наука", 1971.

[6]. Sutherland, Ivan E., Micropipelines.

[7]. Tiberiu Chelcea, Girish Venkataramani, Seth C. Goldstein, SelfResetting Latches for Asynchronous MicroPipelines, Proceedings of the 44th annual ACM/IEEE Design Automation Conference, June 2007.

[8]. Mannakkara, C., Yoneda, T., Asynchronous pipeline controller based on early acknowledgement protocol, National Institute of Informatics: DI, Graduate University for AS, Tokyo, Japan, NII-2009-015E, Sept. 2009.

[9]. GALAXY-Project, Milos Krstic, , Specification of optimized GALS interfaces and application scenarios, GALAXY, 12-2008.

[10]. Jens Muttersbach, Globally asynchronous locally synchronous, architecture for VLSI Systems, PhD thesis [PDF], ETH Zurich, Diss. ETH №14155, 2001.

[11]. Milos Krstic, Eckhard Grass, New GALS Technique for Datapath Architectures, in Integrated circuit and system design: power and timing modeling, by Jorge Juan Chico, Enrico Macii, p.161, books.google.com, 2003; Lecture Notes in Computer Science, ISSN 0302-9743, Volume 2799/2003.

[12]. Eckhard Grass, Frank Winkler, Miloš Krsti, Enhanced GALS Techniques for Datapath. Applications.

[13]. Xin Fan, Miloš Krstić, Eckhard Grass, Analysis and Optimization of Pausible Clocking based GALS Design.

[14]. Kenneth Yun, Peter A. Beerely, Julio Arceo, High-Performance Asynchronous Pipeline Circuits, In Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems, IEEE Computer Society Press, 1996.

[15]. Ran Ginosar, Fourteen ways to fool your synchronizer - Asynchronous Circuits, IEEE Pr. of the 9th ASYNC'03.

[16]. Stefan Hirschmann, Asynchronous processors, Seminar Embedded System Design, Institute of Computer Science, University of Innsbruck, February 25, 2008.

[17]. Chang-Jiu Chen, Wei-Min Cheng, Hung-Yue Tsai, Jen-Chieh Wu, A quasi-delay-insensitive microprocessor core Implementation for Microcontrollers, Journal of information science and engineering 25, 543-557 2009.

[18]. Montek Singh, Chapel Hill, Steven M. Nowick, US Patent, 2005/0156633, Circuits and methods for high-capacity asynchronous pipeline processing.

[19]. Ronald J. Tocci, Neal S. Widmer, Digital systems: principles and applications, 8th ed., Prentice-Hall Inc., ISBN 0-13-085634-7, 2001.

[20]. Pong P. Chu, RTL Hardware Design Using VHDL: Coding for Eflciency, Portability and Scahbility, Wiley IEEE Press, ISBN-13: 978-0-471-72092-8, 2006.

[21]. Daniel Page, Practical Introduction to Computer Architecture, Springer, ISBN 978-1-84882-255-9, 2009.

[22]. Richard F. Tinder, Asynchronous Sequential Machine Design and Analysis, Morgan and Claypool, ISBN: 9781598296907, 2009.