ISSN 1292-862

TIMA Lab. Research Reports

QDI Latches Characteristics and Asynchronous Linear-Pipeline Performance Analysis

YAHYA E.**, RENAUDIN M.*

* TIMA Laboratory, 46 avenue Félix Viallet 38000 Grenoble France
** Benha High Institute of Technology, Benha, Egypt

TIMA-RR--06/06-03--FR

TIMA Laboratory,
46 avenue Félix Viallet, 38000 Grenoble France

QDI Latches Characteristics and Asynchronous Linear-Pipeline Performance Analysis

Eslam Yahya[1,2] and Marc Renaudin[1]

[1] TIMA Laboratory, CIS Group
46 Av. Félix Viallet, Grenoble, France
{Eslam.Yahya, Marc.Renaudin}@Imag.fr
[2] Benha High Institute of Technology, Benha, Egypt

Abstract. Asynchronous logic is a hot topic due to its interesting features of power saving, low noise and robustness to parameters variations. However, its performance analysis is relatively complex. In fact, the handshaking protocol strongly influences the performance of the pipelined architectures. This paper introduces verified Standard-Logic schematics for QDI asynchronous latches and analyzes their characteristics. Buffering capacity and protocol gain are defined and analyzed. Based on this analysis, reduced performance equations are first introduced. By means of the dependency graphs, a new formal method is then proposed to analyze the performance of asynchronous linear-pipeline. This methodology is used to derive general equations for each latch type. Contrarily to previously proposed methods, this method can be applied to any asynchronous linear-pipeline without restrictions on its functional block delays. Therefore, the contributions of this paper enable the designers to understand the benefits brought by the different asynchronous latches, to compare them and make the right choice according to their design constrains. …

1 Introduction

Integration capacity of the recent chips is gradually increasing. This increasing introduces many problems especially in power consumption, robustness against environment-parameters variations and Electro Magnetic emission. Asynchronous circuits seem to be a practical solution for such problems [Mar 03], [Pet 02]. Research and industry are increasingly motivated towards the asynchronous circuits due to the following interesting features [Jac 03], [Jen 01], [Van 99]: (1) Low power consumption, (2) No global-signal distribution problems, (3) High security, (4) Low emitted EM noise and (5) Tolerance against environment parameters change. In case of asynchronous circuits, latches are complex compared to the synchronous Latches/Registers. Asynchronous latches not only memorize the data, but also implement the communication protocol between different system-blocks. As a result, the system performance is strongly affected by the buffering protocol. The exact performance equations for the latches are essentials but never had been derived before. This paper discusses this idea in the context of linear-pipeline. Non-linear pipeline will be considered in another paper. Fig.1 shows the structure of asynchronous linear-pipelines used in the paper.

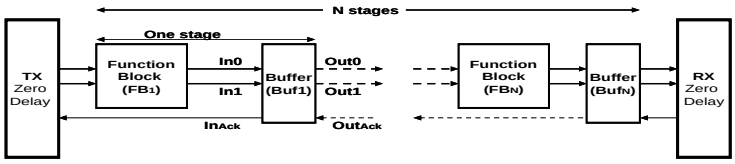

Fig. 1. Structure of an Asynchronous Linear-Pipeline

We faced difficulty in reading many papers concerning the asynchronous logic because the used conventions are not clearly defined. This paragraph defines the conventions used in this paper. The word buffer will be used instead of latch for consistency with the other literatures. The *Linear-Pipeline* can be defined as "simple pipeline structure that contains no fork or join". It can contain any number of stages, each *Stage* consists of "One function block, which is transparent to the acknowledgment signals, and one asynchronous buffer". The *Function Block* "Is composed of logic elements those perform the required processing". Asynchronous buffer has two channels, input channel and output channel. The *Channel* used here is a dual rail channel, which expresses each bit using two wires (0=01, 1=10, Reset=00) and a third wire is used for the acknowledgment. After Reset, all outputs hold Reset value '00', and the acknowledgment is high. The buffer at $stage_i$ 'Buf_i' receives its input from function block 'FB_i' and sends its output to block 'FB_{i+1}'. The communication protocol used in this paper is the four-phase protocol, which is a *Return-to-Zero protocol* (RTZ) [Jen 01], [Mar 03]. In RTZ protocols, the individual *Data Pattern* consists of two tokens. The *Evaluation Token* "which expresses the data (01 or 10)" and *Reset Token* "which resets the logic to prepare it for the next evaluation". The token becomes a *Bubble* if the Acknowledgement signal, confirming that the following stage does not need the token any more, is received. *Buffer Cycle Time Pn* is "The time needed by the buffer at $stage_n$ to process a complete data pattern". The starting point of the pipeline is the *Transmitter* (TX), where the ending point is the *Receiver* (RX). TX and RX are considered zero-delay, which means that TX produces a new token as soon as it has the acknowledgment and RX produces the acknowledgment as soon as it has a new token. This assumption isolates the pipeline from the environment effects. There are plenty of asynchronous buffers [Ste 96], [And 95]. This paper concerns WCHB (Weak-Conditioned Half Buffer), PCHB (Pre-Charged Half Buffer) and PCFB (Pre-Charged Full Buffer). These buffers are originally introduced as pre-charged buffers [And 95]. However, their handshaking protocols can be used in any circuit regardless the logic type. The same naming conventions are used in this article because they are well known, although the proposed architectures are based on standard gates not pre-charged gates. We suggest changing these names to express the protocol regardless the implementation scheme. We reviewed the schematics of these buffers and realized that verified schematics are in need. A complete design and verification process is proposed in [Esy 06]. The paper is structured as two main parts; the first part introduces a verified schematic for each buffer type, discusses the different characteristics of each buffer and proposes reduced performance equations based on structural analysis. The second part discusses the formal methods of asynchronous buffer performance-analysis and proposes a new method that produces general equations those can be applied to any asynchronous linear-pipeline.

2 Buffering Capacity

Buffering capacity can be defined as "The number of cascaded tokens the buffer can simultaneously memorize". Tokens here mean the unacknowledged tokens, and cascaded tokens can be in any order (Evaluation + Reset or Reset + Evaluation). The following figures show the verified schematics for the three buffers.

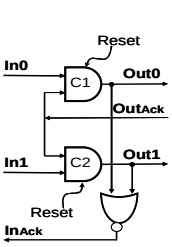

Fig. 2. WCHB

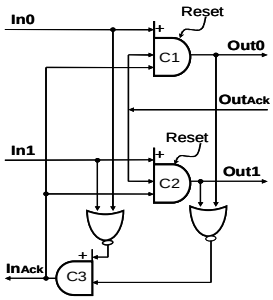

Fig. 3. PCHB

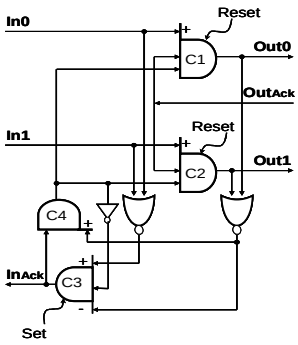

Fig. 4. PCFB

2.1 WCHB Capacity and PCHB Capacity

Fig. 2 and Fig. 3 show WCHB and PCHB circuit diagrams respectively. The two Muller gates C1 and C2 in both buffers can only hold either an Evaluation token or a Reset token at a time. After receiving the acknowledgment signal on the output side, the token becomes a bubble and the buffer is ready to memorize another token. Consequently, WCHBs and PCHBs have a buffering capacity of one token. They can only memorize half of the data pattern, so they are called half buffers.

2.2 PCFB Capacity

PCFB circuit appears in Fig. 4. In all papers, this buffer is proposed as a full buffer, which is not actually accurate. Analyzing the circuit diagram shows that PCFB is something between half buffer and full buffer. Suppose that the buffer Buf_i is empty, it receives an evaluation token, memorizes the token inside the Muller gates C1 and C2, and then responds by putting the In_{Ack} low. That means the outputs of C3 and C4 are low. If FB_{i+1} is slower than FB_i then, FB_i sends the reset token. This makes C3 going high giving In_{Ack} high, which means the channel is free and FB_i can send the next token. Now the buffer is memorizing the evaluation token in (C1 and C2) and memorizing the reset token by two things, the low output of C4 and the high output of C3. In that case, PCFB is memorizing the two tokens, Evaluation then Reset, simultaneously. Thus, this buffer now behaves as a full buffer. Suppose now, FB_{i+1} acknowledges the memorized Evaluation token by putting Out_{Ack} low, C1 and C2 go low, which makes C4 going high. Because C3 is high, FB_i sees the channel free and it sends a new Evaluation token. Now the buffer memorizes only one token, the reset token, and it

has unacknowledged Evaluation token at its inputs. Can the buffer acknowledge this new token as it is expected to do? As a full buffer the answer should be yes, but the real answer is no. This is because it has no real memory elements at its inputs. The conclusion is that, PCFB behaves as a full buffer when it memorizes an Evaluation token followed by a Reset token. However, it behaves as a half buffer when it memorizes a Reset token followed by an Evaluation token. Hence, PCFB sometimes has a buffering capacity of two tokens, other times it has a buffering capacity of one token.

3 Protocol Gain

Each asynchronous buffer defines a different protocol for handling the relation between two function blocks. What is meant by *Protocol Gain* is "How much is the gain in terms of the total delay required to complete the processing of one input data pattern". Analyzing the behavior of any buffer shows that, at $stage_i$, Buf_i receives data from FB_i, memorizes this data, sends the data to FB_{i+1} and acknowledges FB_i. This behavior is ruled by two facts. The first fact is that, Buf_i cannot accept a new token until FB_{i+1} acknowledges the previous token. The second fact is that, Buf_i cannot out a new token until this token is sent by FB_i. These two facts are logical, but they enforce some sequential relations between the buffer sides. How can we break these two facts, or one of them, to add some concurrency between the two sides? The first fact can be easily, but costly, broken, simply by adding more buffering capacity to Buf_i. The more buffering capacity we add, the more Buf_i can accept tokens which have no place at FB_{i+1}. What about the second fact, the answer lies in the question, what does Buf_i expect from FB_i? It expects data pattern that consists of two cascaded tokens, Evaluation and Reset. Because Buf_i cannot predict what evaluation token will be sent by FB_i, then the sequential relation here is obligatory. However, if it receives an evaluation token, it knows that the coming token is a Reset one even before FB_i sends it. Here we can gain some concurrency if Buf_i generates the Reset token for FB_{i+1} before receiving it from FB_i. Subsequently, two kinds of gain can be defined. The first one is the *Extra Buffering Gain* (EBG) "Which results from adding more buffering capacity to the buffer". The second type is the *Token Expectation Gain* (TEG) "which appears when Buf_i is able to generate the Reset token for FB_{i+1} before receiving it from FB_i".

3.1 Gain of Each Buffer Type

WCHB is a simple protocol that adds a single memory space between two-cascaded blocks. As a result, FB_i is enabled to give its current token to Buf_i and starts a new processing with the next token (EBG). If the function blocks have unequal Evaluation and Reset times, then the total delay of the system using WCHB is twice the worst of them. This is because even if FB_i finished its Evaluation/Reset before FB_{i+1} finishes its Reset/Evaluation; it should wait FB_{i+1} to finish and vice versa. Consequently, we have:

$$P_{WCHB} = 2 * MAX [T_{Eval}, T_{Reset}]. \quad \textbf{(1)}$$

By recalling Figure 3, it is obvious that PCHB has more concurrency than WCHB. How can we exploit this concurrency? Suppose that the function blocks have reset time larger than evaluation time. Hence, FB_{i+1} finishes its Evaluation and waits for the Reset token. Here the key advantage of PCHB appears. It generates the reset token for FB_{i+1} causing an overlapping between the reset phases of the both sides (TEG). This reduces the total delay from twice the worst delay as WCHB to the summation of Evaluation time and Reset time. On the other hand, if the Evaluation time is the largest time, then PCHB performance will be the same as WCHB. Therefore, we have:

$$P_{PCHB} = T_{Eval} + MAX [T_{Eval}, T_{Reset}]. \quad \textbf{(2)}$$

PCFB has the ability to generate the Reset token for FB_{i+1} before receiving it from FB_i (TEG). In addition, it can memorize two tokens at a time with the order mentioned before (EBG). Hence, its performance gains from the unequally Reset and Evaluation times not only when the Reset time is larger but also when the Evaluation time is.

$$P_{PCFB} = T_{Eval} + T_{Reset}. \quad \textbf{(3)}$$

3.2 Simulation Results of Zero-Delay Buffers

A test circuit, as in Fig. 1, is designed using the different buffer types and identical function blocks in all stages. Its simulation results are compared with the expected results from the equations. This is done in order to insure the consistency of the equation with the real circuit performance. The buffer internal delays are neglected in order to concentrate on protocols differences regardless the differences in circuitry. Table 1 summarizes the simulation results and shows that they are coherent with equations (1), (2) and (3). It is obvious from the table that, WCHB has the same performance in both cases. PCHB gains when $T_{Eval}<T_{Reset}$. However, PCFB gains in both cases.

Table 1. Simulation results of Zero-Delay Buffers

$T_{Eval}<T_{Reset}$	WCHB	PCHB	PCFB	$T_{Eval}>T_{Reset}$	WCHB	PCHB	PCFB
100/300	600	400	400	300/100	600	600	400

4 Formal Performance-Analysis

Although the above equations give good estimation of the buffers performance, more formal and detailed equations are needed. Designers need exact equation to choose the suitable buffer for their circuits. One of the formal methods that can be used is the *Dependency Graph* method. It is a graph-based method where the nodes of the graph correspond to specific rising or falling transitions of the circuit components, and the arcs depict the dependencies between signals transitions [Ted 94], [Ted 90]. If all the stages have the same function blocks, the graph can be folded. Buffers Unfolded and Folded dependency-graphs are shown in Fig. 5.

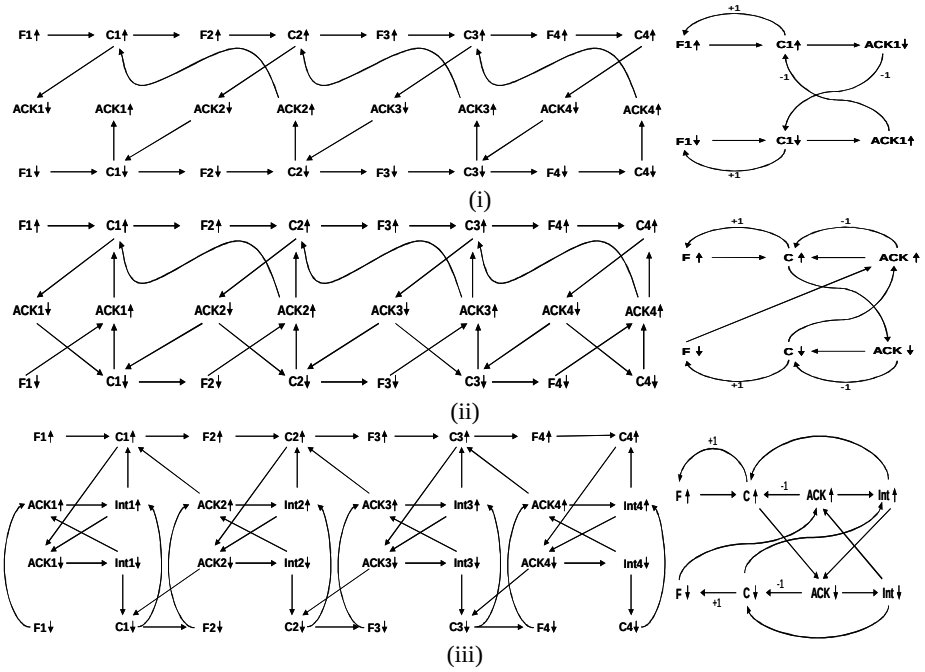

Fig. 5. (i)WCHB, (ii) PCHB and (iii) PCFB Unfolded/Folded Dependency Graphs (C: Buffer output, F: Function-Block output, Ack: Acknowledgment Signal, ↑/↓: UP/Down Transitions)

5 Performance-Analysis Using Folded Dependency-Graphs

This method is based on the folded graphs, and can be summarized into two steps. From the Folded dependency graph, classify all loops in which the summation of stages-indices (represented by I) is less than, greater than and equal to zero in three subsets. After that, obtain the Cycle Time "P" as the Max Latency of loops combinations in which (I = 0). More details can be found in [Ted 94], [Ted 90].

5.1 Buffers Performance-Analysis Using Folded Dependency-Graphs

WCHB folded dependency-graph is shown in Fig.5.i. From this figure, the loops with index-sum greater than Zero are, [C↑, F↓], [C↓, F↑], their index-sum is (+1). Concerning the loops with index-sum less than Zero, there is only one loop [C↑, Ack↓, C↓, Ack↑] with index-sum of (-2). There is no loop with index-sum of Zero; it will be a composition of the other loops. WCHB equation is shown in eq.4. Note that, the second part of the equation is multiplied by 2 to obtain total index sum of Zero. Applying the assumption of zero-delay buffer ($T_{C\uparrow} = T_{Ack\downarrow} = T_{C\downarrow} = T_{Ack\uparrow} = 0$) where $T_{F\uparrow}/T_{F\downarrow}$ are T_{Eval}/T_{Reset} respectively, then the exact equation (4) confirms the estimated one (1).

$$P_{WCHB} = (T_{C\uparrow} + T_{Ack\downarrow} + T_{C\downarrow} + T_{Ack\uparrow}) + 2\ \mathrm{Max}\ [\ (T_{C\uparrow} + T_{F\downarrow}), (T_{C\downarrow} + T_{F\uparrow})\ ]. \quad \textbf{(4)}$$

The folded graph of PCHB is shown in Fig.5.ii. Due to its concurrency; the extraction of the loops subsets is more complicated. The possible loops in this circuit are (F↑, C↑) with index-sum of (+1), (F↓, Ack↑, C↑, Ack↓, C↓) with index-sums of (+1, 0, -1) and (C↑, Ack↓, C↓, Ack↑) with index-sums of (0, -1, -2). There are some ambiguity in extracting the loops and assigning their indices sums. The last two loops have three index-sums, which is ambiguous especially when the cycle time is being calculated. For example, the last loop index sum depends on the chosen path between (Ack↓, C↓) and between (Ack↑, C↑). If the zero arcs are chosen, then the total sum is zero. On the contrary, if one of them is zero and the other is (-1), the total sum is (-1). Finally, if both paths are through the (-1) arcs then the resultant sum is (-2). It is ambiguous for us to calculate the cycle time using the above loops and very hard to formulate an equation to express the cycle time.

Concerning the PCFB circuit, it has an internal memorized signal; it is the output of C4 in Fig. 4. We called this signal (Int). It appears in the folded graph in Fig 5.iii. The possible loops of this buffer are, (F↑, C↑), (F↓, Ack↑, Int↑, Ack↓, Int↓, C↓) with index of (+1) and (Ack↑, C↑, Ack↓, Int↓), (Ack↑, C↑, Ack↓, C↓, F↓), (Ack↓, C↓, Int↑), (Ack↓, C↓, Int↑, C↑), with index of (-1). Finally, the loop (F↓, Ack↑, Int↑, Ack↓, C↓) has an index of (0). The resultant cycle time equation is shown in eq.5. Where eq.6. shows the cycle time after applying the zero-delay buffer assumption. Eq.6. is not the same as eq.3. In addition, neither eq.5 nor eq.6 can explain the performance gain in PCFB when TEval is lower than TReset.

$$P_{PCFB}=\text{Max}\,[(T_{F\downarrow}+T_{Ack\uparrow}+T_{Int\uparrow}+T_{Ack\downarrow}+T_{C\downarrow}),(\text{Max}[(T_{F\uparrow}+T_{C\uparrow}),(T_{F\downarrow}+T_{Ack\uparrow}+T_{Int\uparrow}+T_{Ack\downarrow}+T_{Int\downarrow}+T_{C\downarrow})]+\text{Max}[(T_{Ack\uparrow}+T_{C\uparrow}+T_{Ack\downarrow}+T_{Int\downarrow}),(T_{Ack\uparrow}+T_{C\uparrow}+T_{Ack\downarrow}+T_{C\downarrow}+T_{F\downarrow}),(T_{Ack\downarrow}+T_{C\downarrow}+T_{Int\uparrow}),(T_{Ack\downarrow}+T_{C\downarrow}+T_{Int\uparrow}+T_{C\uparrow})])]. \quad \textbf{(5)}$$

$$P_{PCFB} = \text{Max}\,[T_{F\uparrow}, T_{F\downarrow}] + T_{F\downarrow}. \quad \textbf{(6)}$$

6 Performance-Analysis Using Unfolded Dependency-Graphs

The *Total Cycle Time* P_{Total} is "The total time the circuit takes to completely process a complete data pattern". Hardness of calculating the cycle time comes from concurrency of the circuit events. However, in any circuit, regardless how much its events are concurrent, there is an event-transition where all the circuit events are requisites. As a result, this event-transition directly or indirectly synchronizes all the other events; which insures that, all parallel event-sequences are allowed to start after this transition. We call such a transition, the *Main Synchronization Event-Transition*. Starting from this transition and tracing the circuit until returning back, while considering the longest of any parallel event-sequences, should give the correct cycle time. This section proposes a methodology based on the unfolded-graphs, this methodology can be summarized as: 1) Draw the unfolded-graph of your circuit. 2) Define the main synchronization event-transition. 3) From this transition, going up for example, trace the circuit until reaching the dual transition considering the longest of any parallel event-sequences. This forms the first part of the equation. 4) Returning back to the main synchronization event-transition, considered in step 3, while considering the longest of

any parallel event-sequences, forms the second part of the equation. Note, *starting and ending event-transitions should be considered once.* Because the ending transition of the equation first part is the starting transition of its second part etc. then, either starting or ending transition is considered every time. This prevents from adding the same transition twice. Contrarily to the other method, this method has no restriction on the function blocks. Each stage can have a different function block with different delay values, which makes the methodology a realistic solution for linear pipeline. The following subsections apply this method to each buffer type to extract a general stage equation (Pn). In all of these equations 8, 9 and 10, the performance of any stage depends on the previous, current and next stages. Subsequently, the total cycle time will be the maximum of these stage equations and it can be obtained by the general equation of eq.7. Where n, is the stage index, N is the total number of stages and Pn is the general stage equation of each buffer in the linear pipeline.

$$P_{Total} = Max_{(n=1 \text{ to } n=N)} [\ Pn]. \quad \textbf{(7)}$$

6.1 Buffers Performance-Analysis Using Unfolded Dependency-Graphs

WCHB unfolded-graph is shown in Fig.5.i. The main synchronization event lies in the outputs of C1&C2 in Fig.2. This event appears in the graph as C↑&C↓. Starting from C↑, we write down the first part of the equation, Max [(F↑,C↑,Ack↓,C↓),(Ack↓,C↓,F↓, C↓)]. C↑ is considered as the starting transition and we choose not to include it while including the ending transition. It is important to note that we navigate from C↑ of a stage to C↓ in the same stage. From C↓, we return back to C↑ for writing the equation second part, Max [(F↓, C↓, Ack↑, C↑), (Ack↑, C↑, F↑, C↑)]. The final cycle time equation is as follows:

$$Pn_{WCHB}=Max[(T_{F(n+1)\uparrow}+T_{C(n+1)\uparrow}+T_{Ack(n+1)\downarrow}+T_{C(n)\downarrow}),(T_{Ack(n)\downarrow}+T_{C(n-1)\downarrow}+T_{F(n)\downarrow}+T_{C(n)\downarrow})] \quad \textbf{(8)}$$
$$+Max[(T_{F(n+1)\downarrow}+T_{C(n+1)\downarrow}+T_{Ack(n+1)\uparrow}+T_{C(n)\uparrow}), (T_{Ack(n)\uparrow}+T_{C(n-1)\uparrow}+T_{F(n)\uparrow}+T_{C(n)\uparrow})].$$

In fact, this equation is equivalent to eq.4 if all stages have similar buffers. By applying the zero-delay buffer assumption, this equation gives eq.1.

The new procedure is applied on PCHB unfolded-graph, Fig.5.ii. The output transitions of Muller gates C1&C2 are the main synchronization event and C↑ is the starting transition. This gives the cycle time equation appears in eq.9. The assumption of zero-delay buffer makes eq.9 the same as eq.2. By using the new method, this is the first time we have performance equation for PCHB.

$$Pn_{PCHB} =Max[(T_{F(n+1)\uparrow}+T_{C(n+1)\uparrow}+T_{Ack(n+1)\downarrow}+T_{C(n)\downarrow}),(T_{Ack(n)\downarrow}+T_{C(n)\downarrow})]+ \quad \textbf{(9)}$$
$$Max[(T_{F(n+1)\downarrow}+T_{Ack(n+1)\uparrow}+T_{C(n)\uparrow}),(T_{Ack(n)\uparrow}+Max[(T_{C(n-1)\uparrow}+T_{F(n)\uparrow}+T_{C(n)\uparrow}),(T_{C(n)\uparrow})])].$$

Concerning PCFB unfolded dependency graph, Fig.5.iii, the same procedure is applied while taking the main synchronization event at the outputs of C1&C2 and C↑ is the starting transition, which gives eq.10. Unlike eq.5, the zero-delay buffer assumption makes eq.10 similar as eq.3.

$$Pn_{PCFB} = Max[(T_{F(n+1)\uparrow}+T_{C(n+1)\uparrow}+T_{Ack(n+1)\downarrow}+T_{C(n)\downarrow}),(T_{Ack(n)\downarrow}+T_{Int(n)\downarrow}+T_{C(n)\downarrow})] \quad \textbf{(10)}$$
$$+Max\ [\ (T_{F(n+1)\downarrow} + T_{Ack(n+1)\uparrow} + T_{C(n)\uparrow}), (T_{Int(n)\uparrow} + T_{C(n)\uparrow})\ ].$$

6.2 Simulation of Assumed-Delay Buffers with Random Function-Block Delays

A ten-stage linear-pipeline circuit, like in Fig.1, is simulated. Buffers are modeled in VHDL using assumed internal delays. These delays are around the expected delays of a 130 nm cmos process. Using the assumed delays, Table 2 shows the values of the equation-parameters for each buffer type. Function block delays are randomly chosen to confirm the generality of the proposed method. Table 2 shows these values with the total cycle time calculated from the equations and obtained from simulation.

Table 2. Buffer Equation-Parameters, Function-Block Delays and Total Cycle Times

WCHB		PCHB		PCFB		
T_C=150	T_{Ack}=30	T_C=200	T_{Ack}=130	T_C=200	T_{Ack}=160	T_{Int}=130
Stage:T_{Eval}/ T_{Reset}	1: 500/900	2: 2/1.5 ns	3: 300/100	4: 10/1 ns	5: 5/6 ns	
Stage:T_{Eval}/ T_{Reset}	6: 60/100	7: 2/9 ns	8: 1/12 ns	9: 7/4 ns	10: 390/130	
WCHB Total Cycle Time	PCHB Total Cycle Time			PCFB Total Cycle Time		
Eq/Sim: 21.66/21.66 ns	Eq/Sim: 16.06/16.06 ns			Eq/Sim: 13.92/13.92 ns		

From the above table, it is obvious that the equation results are coherent with the real performance. Moreover, these equations can generally be applied to linear-pipelined architectures. Fig.6 A, B and C show the stage cycle times for WCHB, PCHB and PCFB respectively. It is clear from the figure that no buffer type is the superior in all cases. For example in stages 1 and 3, WCHB is the best buffer. In stage 7, PCHB is the best solution. On the other hand, PCFB is superior to the others in many cases.

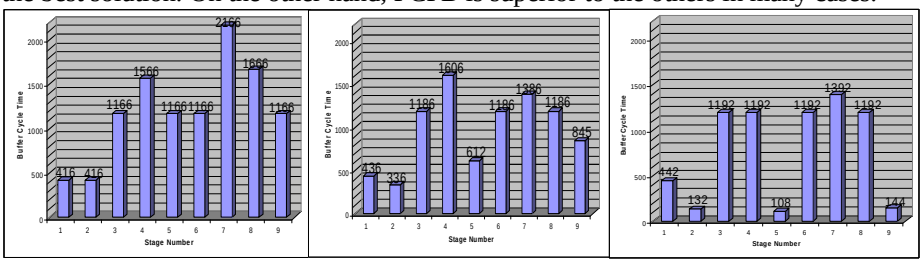

Fig.6. (A)WCHB, (B)PCHB and (C)PCFB Stage Cycle Times, the time unit is ten-picoseconds

7 Conclusion and Future Work

This paper has introduced verified standard-logic schematics for WCHB, PCHB and PCFB. As far as we know, these are the first schematics in standard logic for these buffers, which enables the use of standard libraries instead of special ones. The buffering capacity of each type is defined and the variable capacity of PCFB is explained. Moreover, this paper proves that PCFB is not really a full buffer as mentioned in all previous works. The protocol gain is defined and classified into EBG and TEG, these parameters are useful guidelines for designers to understand how to compare between buffering protocols and choose the suitable one according to their circuit conditions. Formal performance analysis using the folded-graph method has been done. It is found that this method is not suitable because it gives some ambiguity and even incorrect

results. Moreover, this method is not generic; it is only applicable on linear pipeline with similar function blocks in all stages. The unfolded-graph method is proposed then used to derive general performance equations for each buffer type. Obviously, this method is working with all tested buffers. In addition, it is applicable to any linear pipeline with no restriction on the delays of the function blocks. Such general performance equations are new contributions in the area of performance analysis. WCHB performance is the worst of the all, where it cannot gain from the unbalanced Evaluation and Reset times of the function blocks. PCHB gains in case of longer Reset time. PCFB is the superior, where its performance gains in case of unbalanced Evaluation and Reset times regardless which is the longest.

An important conclusion is that, no buffer type is absolutely the best; it depends on the circuit conditions, if all functional blocks are identical with symmetric Evaluation and Reset times, then WCHB is the best. However, PCHB will be better if the Reset is longer in all blocks. PCFB is recommended when the evaluation is longer, but it works also with longer Reset. In case of randomized block delays, PCFB is expected to achieve the best results in terms of the total delay. However, mixing between the different types can achieve the best results in terms of area as well as performance, but it should be done very carefully. The general equations introduced in this article constitute a theoretical basis for design automation. If these equations are integrated inside a tool like our TAST tool, then the tool can automatically select the best buffer type according to the circuit conditions. More study for mixing between different types is in progress. In addition, extending these equations to be applied to nonlinear pipeline is in the phase of study.

References

[And 95] Andrew Lines: Pipelined Asynchronous Circuits. Master Thesis (1995)

[Esy 06] Eslam Yahya, Marc Renaudin: Asynchronous Buffers Characteristics: Molding and Design. Research Report, TIMA-RR--06/-01--FR (2006)

[Jac 03] Jacques J.A., Simon Moore, Huiyun Li, Robert Mullins, George Taylor: Security Evaluation of Asynchronous Circuits. CHES, pp. 137-151 (2003)

[Jen 01] Jens SparsØ, Steve Furber: Principles of Asynchronous Circuit Design. A System Perspective. Kluwer Academic Publishers (2001)

[Mar 03] Marc Renaudin, Joao Leonardo: Asynchronous Circuits Design: An Architectural Approach. (2003)

[Pet 02] Peter A. Beerel: Asynchronous Circuits: An Increasingly Practical Design Solution. Proceedings of ISQED'02, IEEE Computer Society (2002)

[Ste 96] Stephen. B. Furbe and Paul Day: Four-Phase Micropipline Latch Control Circuits. IEEE Transitions on VLSI Systems, 4(2):247-253, (1996)

[Ted 90] Ted Williams: Latency and Throughput Tradeoffs in Self-Timed Speed-Independent Pipelines and Rings. Technical Report No. CSL-TR-90-431 (1990).

[Ted 94] Ted Williams: Perfoemance of Itrative Computation in Self-Timed Rings. Journal of VLSI Signal Processing, 7, 17-31 (1994).

[Van 99] C.H. (KEES) Van Berkel, Mark B. Josephs, Steven M. Nowick: Scanning the Technology. Applications of Asynchronous Circuits. Proceedings of the IEEE, Vol. 87, No 2 (1999)