ISSN 1346-5597

National Institute of Informatics

NII Technical Report

# Asynchronous Pipeline Controller Based on Early Acknowledgement Protocol

Chammika Mannakkara and Tomohiro Yoneda

NII-2008-009E
Sept. 2008

URL: http://www.nii.ac.jp/ Tokyo 101-8430, JAPAN +81-3-4212-2000 FAX: +81-3-3556-1916

PAPER

# Asynchronous Pipeline Controller Based on Early Acknowledgement Protocol

**Chammika MANNAKKARA**[†], *Nonmember and* **Tomohiro YONEDA**[†], *Member*

**SUMMARY** A new pipeline controller based on Early Acknowledgement protocol is proposed for bundled-data asynchronous circuits. The Early Acknowledgement protocol indicates acknowledgement by the falling edge of the acknowledgement signal in contrast to the 4-phase protocol which indicates it on the rising edge. Thus, it can hide the overhead caused by a resetting phase of the handshake cycle. Since we have designed our controller assuming several timing constraints, we first analyze the timing constraints under which our controller correctly works, and discuss the appropriateness. Advantages of employing the Early Acknowledgement protocol in a pipeline controller is demonstrated by comparing performance of the proposed controller with that of two other pipeline controllers, namely, a very high-speed 2-phase controller and an ordinary 4-phase controller, both analytically and experimentally. We have obtained interesting results in the case of a non-linear pipeline with a Conditional Branch operation. Since Early Acknowledgement protocol employs return-to-zero control signals like 4-phase protocol, our controller for Conditional Branch operation is simple in construction identically to the 4-phase controller. A 2-phase controller for the same operation needs to have a little complicated mechanism to handle the 2-phase operation due to non-return-to-zero control signals. For such simplicity of the implementation, our controller has a slightly better performance compared to the 2-phase controller when each stage has a processing element. Thanks to the superiority of the Early Acknowledgement protocol, our controller substantially outperforms the 4-phase controller too.

***key words:*** *Asynchronous Pipelines, Early Acknowledgement Protocol, Bundled-Data Asynchronous Circuits*

## 1. Introduction

A host of asynchronous pipeline controllers has been proposed over the years [4, 5, 6, 10, 15, 17]. Mainly they use either 2-phase signalling protocol or 4-phase signalling protocol.

In 2-phase or transition signalling protocol, *events* (requests and acknowledgements) are identified at a transition of the control signals either from low-to-high or high-to-low, and the levels of control signals have no significance. Hence, as shown in Fig. 1(a), a whole request-acknowledge cycle is completed when *both* signals make the same transition from one state to the other. The MOUSETRAP [4], a simple and robust linear pipeline controller, is based on this protocol, which proved to operate on ultra-high speeds. However, when using transition signalling protocol, usually translations from 2-phase to 4-phase are required at some points, because in many cases, environment circuits use level sensitive controls.

In the 4-phase protocol as shown in Fig. 1(b), a given cycle has two phases, the working phase and the resetting phase. From the rising edge of request to the rising edge of acknowledgement is the working phase where a request is handled and completion is notified. The return-to-zero of both request and acknowledgement signals constitutes the resetting phase. The different sequencing of these 4-phase signalling transitions leads to different controllers for a range of cost and performance options as shown in [6].

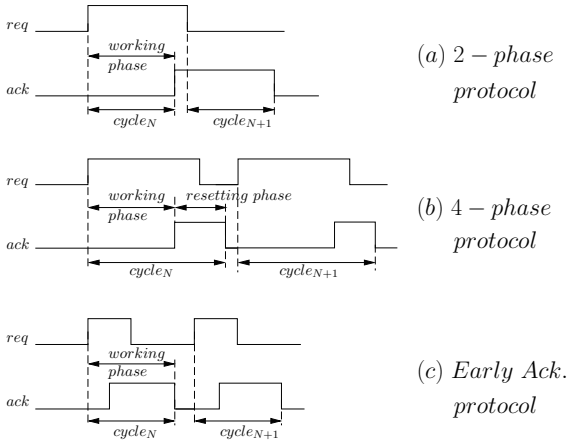


**Fig. 1** Handshake Protocols.

Pipeline controller presented in this paper employs the Early Acknowledgement protocol introduced in [9], where its original idea was presented in [3]. This protocol is an improvement over the simple 4-phase protocol, and can hide the resetting phase of the signalling. In this protocol, the acknowledgement is indicated by the falling edge of the acknowledgement signal whereas in the 4-phase protocol it is indicated with a rising edge. As shown in Fig. 1(c), the acknowledgement signal goes high at any time point when the request signal goes high, thereby allowing the request signal to be reset on an *early acknowledgement*. The actual acknowledgement which is indicated by the falling edge of acknowledgement signal delimits the end of the current transaction and resets the acknowledgement signal for the next request-acknowledge cycle. Hence, this protocol eliminates the resetting phase inherent in the 4-phase protocol and yet retains its simplicity by maintaining the return-to-zero control signals.

In this paper*, we present a new asynchronous pipeline controller based on Early Acknowledgement protocol. A controller based on Early Acknowledgement protocol for

[†]National Institute of Informatics, Graduate University for Advanced Studies, Tokyo, Japan

*This paper is an extended version of [1] published in the ACSD 2008 proceedings

non-linear Conditional Branch operation is also presented. For both cases, we show a set of timing constraints to be satisfied for the proper operation of the proposed controller. These timing constraints are necessary for our controller, because we have designed it using several reasonable timing assumptions in order to simplify the circuit and obtain better performance. Finally, this paper shows the analytical and experimental performance comparison with the existing 2-phase and 4-phase controllers.

The rest of the paper is organized as follows. Section 2 shows the design of our controller and its detailed operation in the case of linear pipelines, as well as the analysis of the timing constraints and performance. The performance comparison to the 2-phase and 4-phase linear controllers is given in Section 3. The design and analysis of Conditional Branch non-linear controller is given in Section 4. Section 5 shows the experimental results for the comparison of three controllers, and Section 6 gives conclusions and future work in the this research.

## 2. Pipeline controller for Early Acknowledgement Protocol

The pipeline controller we present in this paper is an improvement of the controller that we proposed earlier in [2]. The new controller has reduced the overhead of the previously proposed controller under two timing constraints introduced.

### 2.1 Pipeline Operation of Early Acknowledgement Protocol

First, we will define a few naming conventions that we use for Early Acknowledgement controller, 2-phase and 4-phase controllers throughout the rest of this paper. A general diagram of a pipeline using bundled data scheme with logic processing in-between stages is shown in Fig. 2. In the interface of the controller, $Rin_N$ is the request from the input $stage_{N-1}$ and $Ain_N$ is the corresponding acknowledgement signal for the input side. Similarly, $Rout_N$ and $Aout_N$ are the request and acknowledgement to and from the output $stage_{N+1}$. The local clock signal of the stage generated by the controller is $clk_N$.

The logic processing delay ($t_{logic}$) between stages of the pipeline is accounted for by the worst-case matched delay ($t_{MD}$) inserted in the request line between stages. For 2-phase protocol the delay can be symmetric such as a string of buffers, where as for 4-phase protocol (hence, for Early Acknowledgement protocol as well) the delays are asymmetric with a quicker resetting time as shown in Fig. 3. $t_{MD\uparrow}$ represents the variable delay for the rising transition and $t_{MD\downarrow}$ represents the delay for the falling transition. According to our implementation $t_{MD\downarrow}$ equals to $t_{AND\downarrow}$.

Fig. 4 shows the operational waveforms of the Early Acknowledgement controller. As explained in the previous section, in the case of Early Acknowledgement protocol, we use the falling edge of acknowledgement signal to indicate the completion of working phase. Hence, the data $D$ on $data_N$ will be captured to $stage_N$ at the falling edge of $Ain_N$ (i.e., $clk_N = \overline{Ain_N}$), which implies data is expected to be ready on the falling edge of the $Rin_N$. The captured data on $data'_N$ will become valid at $data_{N+1}$ after the processing in-between the two stages. In order for the next stage of the pipeline ($stage_{N+1}$) to be able to capture this valid $data_{N+1}$ by the falling edge of $Ain_{N+1}$, $Rin_{N+1}$ is properly delayed by a delay element $MD$. That is, $t_{MD}$ is determined according to the processing delay $t_{logic}$. Note that it can be observed that the overhead of the controller (i.e., the transition times for $Rout$, $Aout$, and so on) can be entirely hidden in the required matched delay ($t_{MD}$), provided that the processing delay is greater than the controller overhead.

### 2.2 Controller Operation

Fig. 5 depicts the controller that we propose for Early Acknowledgement protocol. We need to adjust the resetting time of *rst* signal with an asymmetric delay $RD$. The implementation of this delay is shown in Fig. 6. $t_{RD\downarrow}$ is the delay that we need, and $t_{RD\uparrow}$ is just equal to $t_{OR\uparrow}$.

Fig. 7 shows the operation of the controller which confirms to the pipelined operation of Fig. 4. Initially, all the control signals are low except for *clk* signal. When the input stage raises the request *Rin*, the controller immediately acknowledges the request by raising *Ain*. At first, this is made possible as there are no pending requests at the output stage through *Rout* (As for the blocked case, see below).

As the acknowledgement is provided by raising *Ain*, *rst* -the input for *A2* AND gate from the asymmetric delay is also raised. When the input stage lowers the request on response the acknowledgement and the data is expected to be ready, the following events occur.

- *Ain* is lowered by the falling of *Rin* thorough *A1*,
- *clk* is raised, latching the new valid data from the input stage to the current stage register, and
- *complete* is raised, generating the rising edge of the output request *Rout*

Once *Rout* is driven high, it can be maintained high

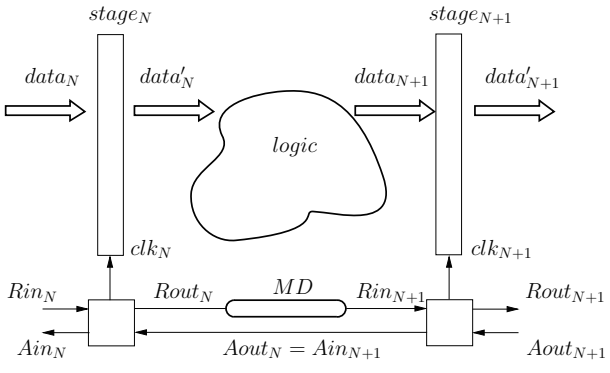


**Fig. 2** A General Pipeline with Logic Processing.

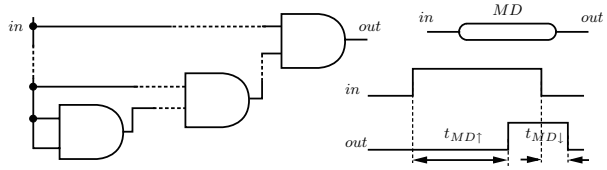


**Fig. 3** Asymmetric Delay for $MD$.

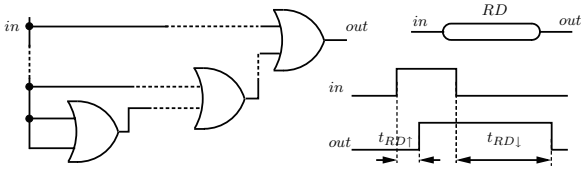


**Fig. 6** Asymmetric Delay for $RD$.

with C-element† even after *complete* signal is lowered by the self-resetting circuit of the controller. This also constitutes a local timing constraint to be satisfied by $t_{RD\downarrow}$ of the self resetting delay i.e., to hold *complete* signal high, long enough to produce *Rout* high before resetting.

Since the controller has fully completed the handshake cycle at the input side, it is free to make a new request on *Rin*. However, as described earlier, the pending output request *Rout* high effectively blocks generating the acknowledgement back to the input side.

Upon receiving acknowledgement high on *Aout*, *Rout* will be lowered, and the blocked request at the input stage will be handled raising *Ain*.

†The C-element used here with a negative input changes its output only when the two inputs have different values, and its output value is equal to that of the positive input.

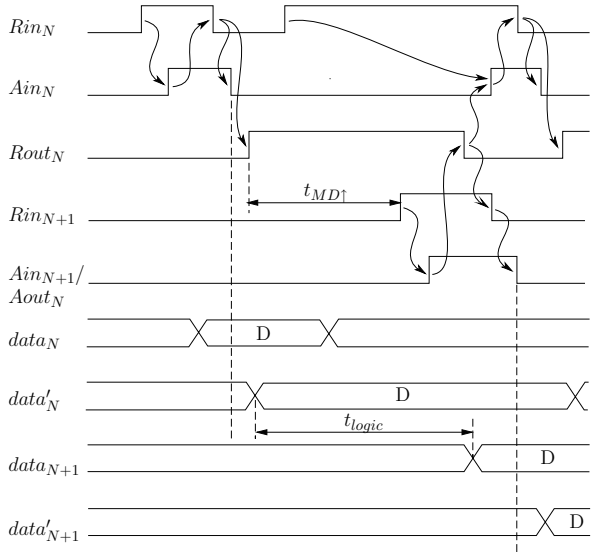


**Fig. 4** Behavior of Early Acknowledgement controller.

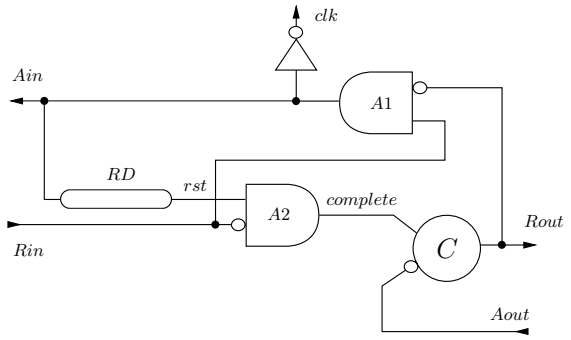


**Fig. 5** Early Acknowledgement pipeline controller.

## 2.3 Timing Constraints

First, we will turn to the timing constraints required for the desired operation as described in Section 2.2. For constraint analysis, we assume that our controller is in a middle stage of a pipeline and its environment (e.g. controllers in the previous and next stages) operates at a speed equal to or slower than our controller. This is because we consider that the linear controller is the fastest, and assuming the environment to be slower than it allows us to evaluate the impact on constraints when more complex operations are built around the linear controller as detailed in Section 4.1. Fig. 8 shows the fastest environment where the delays can be quantified using the controller delays.

We identify two types of expressions throughout the constraint analysis; the constraints and properties. The equation numbers are appropriately prefixed with letter *C* or *P* to distinguish between these types. Constraints are what are *required* to be satisfied where as properties express conditions that *already* hold. We utilize properties of the controller and environment in validating the constraints during our analysis.

**Constraint 1.** The first constraint imposes conditions to prevent data overwriting. As described in the previous section, in the operation of our controller, the pending output request ($Rout$ high) blocks any new requests on $Rin$.

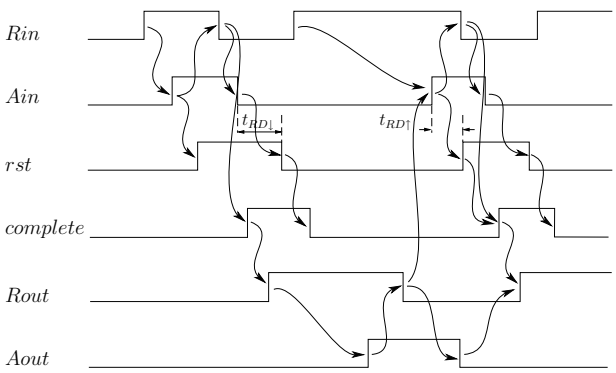


**Fig. 7** Controller Operation.

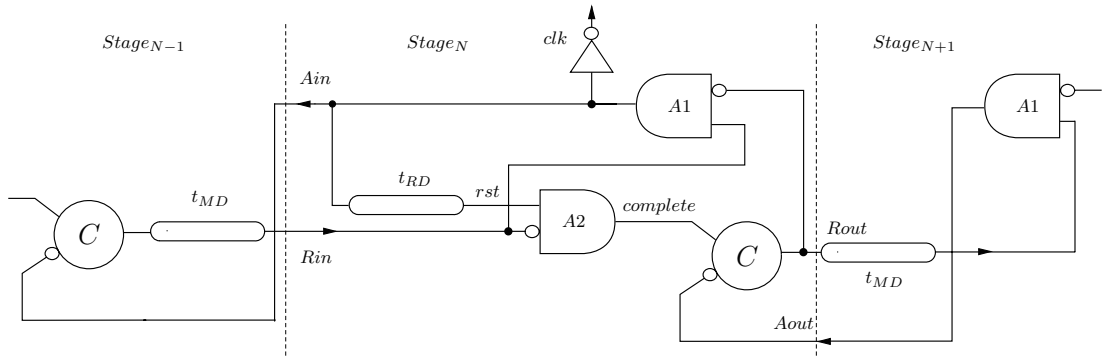


**Fig. 8** Fastest environment for the constraint analysis.

This requires $Rout$ to go high *before* the a new request ($Rin$ high) is received. Thus the timing constraint can be formulated as follows:

$$t_{Rin\downarrow \rightarrow Rin\uparrow} \geq t_{Rin\downarrow \rightarrow Rout\uparrow} \,. \tag{C1}$$

The left-hand-side of the above constraint can be given as:

$$t_{Rin\downarrow \rightarrow Rin\uparrow} = t_{AND\downarrow} + t_{Ain\downarrow \rightarrow Rin\uparrow} \,. \tag{P2}$$

Note that $Ain \downarrow$ is always caused by $Rin \downarrow$ through *A1* AND gate. Since the delays incurred from the environment at the input and output sides are considered to be either equal to or larger than the delays incurred by a linear controller as mentioned previously, the following holds (see Fig. 8).

$$t_{Ain\downarrow \rightarrow Rin\uparrow} \geq t_{C\uparrow} + t_{MD\uparrow} \,. \tag{P3}$$

Thus, (P2) can be rewritten as:

$$t_{Rin\downarrow \rightarrow Rin\uparrow} \geq t_{AND\downarrow} + t_{C\uparrow} + t_{MD\uparrow} \,. \tag{P4}$$

As for the right-hand-side of (C1), we need to consider two cases that different events cause $Rout \uparrow$.

***Case 1:*** If $Aout \downarrow$ is early enough compared to next $Rin \uparrow$, and $Rout \uparrow$ is caused by $complete \uparrow$, the following holds:

$$\begin{aligned} t_{Rin\downarrow \rightarrow Rout\uparrow} =& max(0, -t_{Ain\uparrow \rightarrow Rin\downarrow} + t_{OR\uparrow}) \\ &+ t_{AND\uparrow} + t_{C\uparrow} \,. \end{aligned} \tag{P5}$$

The $max$ operator is used to get the larger of delays from 2 concurrent paths. The first path corresponds to the delays from the input side, and the second path comprises of delays local to the controller in the self-resetting loop. Since the second path is actually originated from $Ain \uparrow$, $t_{Rin\downarrow \rightarrow Ain\uparrow} = -t_{Ain\uparrow \rightarrow Rin\downarrow}$ is used. Again, from the delay assumption of the environment,

$$t_{Ain\uparrow \rightarrow Rin\downarrow} \geq t_{C\downarrow} + t_{AND\downarrow} \tag{P6}$$

holds (see Fig. 8). The occurrence of $t_{MD\downarrow}$ is replaced with equivalent gate delay $t_{AND\downarrow}$ in above environment property. Thus, (P5) can be rewritten as:

$$\begin{aligned} t_{Rin\downarrow \rightarrow Rout\uparrow} \leq& max(0, -(t_{C\downarrow} + t_{AND\downarrow}) + t_{OR\uparrow}) \\ &+ t_{AND\uparrow} + t_{C\uparrow} \,. \end{aligned} \tag{P7}$$

From (P4) and (P7), a conservative version of the constraint (C1) is obtained in the form of constraints for the variable parameter $t_{MD\uparrow}$, the matched delay to be inserted between two stages of the pipeline, as follows:

$$\begin{aligned} & t_{AND\downarrow} + t_{C\uparrow} + t_{MD\uparrow} \geq t_{AND\uparrow} + t_{C\uparrow} \\ & \text{that is,} \qquad t_{MD\uparrow} \geq t_{AND\uparrow} - t_{AND\downarrow} \end{aligned} \tag{C8}$$

and

$$\begin{aligned} t_{AND\downarrow} + t_{C\uparrow} + t_{MD\uparrow} \geq& -(t_{C\downarrow} + t_{AND\downarrow}) + t_{OR\uparrow} \\ &+ t_{AND\uparrow} + t_{C\uparrow} \\ \text{that is,} \qquad & t_{MD\uparrow} \geq t_{AND\uparrow} + t_{OR\uparrow} \\ &- (t_{C\downarrow} + 2 \cdot t_{AND\downarrow}) \,. \end{aligned} \tag{C9}$$

***Case 2:*** If $Aout \downarrow$ is late and causes $Rout \uparrow$, the following holds:

$$\begin{aligned} t_{Rin\downarrow \rightarrow Rout\uparrow} =& - (t_{AND\uparrow} + t_{Ain\uparrow \rightarrow Rin\downarrow}) \\ &+ t_{Rout\downarrow \rightarrow Aout\downarrow} + t_{C\uparrow} \,. \end{aligned} \tag{P10}$$

From the delay assumption (P6), this can be rewritten as:

$$\begin{aligned} t_{Rin\downarrow \rightarrow Rout\uparrow} \leq& - (t_{AND\uparrow} + t_{C\downarrow} + t_{AND\downarrow}) \\ &+ t_{Rout\downarrow \rightarrow Aout\downarrow} + t_{C\uparrow} \,. \end{aligned} \tag{P11}$$

From (P4) and (P11), another conservative version of the constraint (C1) for $t_{MD\downarrow}$ is obtained as follows:

$$\begin{aligned} t_{AND\downarrow} + t_{C\uparrow} + t_{MD\uparrow} \geq& -(t_{AND\uparrow} + t_{C\downarrow} + t_{AND\downarrow}) \\ &+ t_{Rout\downarrow \rightarrow Aout\downarrow} + t_{C\uparrow} \\ \text{that is,} \qquad & t_{MD\uparrow} \geq t_{Rout\downarrow \rightarrow Aout\downarrow} - (t_{C\downarrow} \\ &+ t_{AND\uparrow} + 2 \cdot t_{AND\downarrow}) \,. \end{aligned} \tag{C12}$$

All the constraints derived for $t_{MD\uparrow}$ in cases 1 and 2 (i.e. (C8), (C9) and (C12)) can be satisfied in the preferred application of our controller where there are processing elements within the pipeline and hence the matched delay $t_{MD\uparrow}$ is sufficiently large to meet the above constraints.

**Constraint 2.** The next is a timing constraint to be satisfied by the self resetting delay. *complete* signal should not be self-reset *before Rout* high is produced. This constraint imposes conditions on minimum delay for the self resetting loop $t_{RD\downarrow}$ to satisfy the above condition. We can formulate this constraint as follows.

$$t_{Rin\downarrow \rightarrow complete\downarrow} \geq t_{Rin\downarrow \rightarrow Rout\uparrow} \, . \qquad \text{(C13)}$$

From Fig. 7, the causality relation for $Rin\downarrow$, $Ain\downarrow$, $RD\downarrow$, and $complete\downarrow$ is straight. Thus, the left-hand-side of the above constraint can be given as:

$$\begin{aligned} t_{Rin\downarrow \rightarrow complete\downarrow} &= t_{AND\downarrow} + t_{RD\downarrow} + t_{AND\downarrow} \\ &= t_{RD\downarrow} + 2 \cdot t_{AND\downarrow} \, . \end{aligned} \qquad \text{(P14)}$$

The right-hand-side of the above constraint is the same as that of (C1). Thus, exactly the same two cases as those shown for Constraint 1 are considered, and the following three constraints are obtained for (C13).

***Case 1:*** From (P14) and (P7), a conservative version of the constraint (C13) is obtained as follows:

$$\begin{aligned} & t_{RD\downarrow} + 2 \cdot t_{AND\downarrow} \geq t_{AND\uparrow} + t_{C\uparrow} \\ \text{that is,} \quad & t_{RD\downarrow} \geq t_{AND\uparrow} + t_{C\uparrow} - 2 \cdot t_{AND\downarrow} \end{aligned} \qquad \text{(C15)}$$

and

$$\begin{aligned} t_{RD\downarrow} + 2 \cdot t_{AND\downarrow} &\geq -(t_{C\downarrow} + t_{AND\downarrow}) + t_{OR\uparrow} \\ &\quad + t_{AND\uparrow} + t_{C\uparrow} \\ \text{that is,} \quad t_{RD\downarrow} &\geq t_{OR\uparrow} + t_{AND\uparrow} + t_{C\uparrow} \\ &\quad - (t_{C\downarrow} + 3 \cdot t_{AND\downarrow}) \, . \end{aligned} \qquad \text{(C16)}$$

***Case 2:*** From (P14) and (P11), another conservative version of the constraint (C13) for $t_{RD\uparrow}$ is obtained as follows:

$$\begin{aligned} t_{RD\downarrow} + 2 \cdot t_{AND\downarrow} &\geq -(t_{AND\uparrow} + t_{C\downarrow} + t_{AND\downarrow}) \\ &\quad + t_{Rout\downarrow \rightarrow Aout\downarrow} + t_{C\uparrow} \\ \text{that is,} \quad t_{RD\downarrow} &\geq t_{Rout\downarrow \rightarrow Aout\downarrow} + t_{C\uparrow} \\ &\quad - (t_{AND\uparrow} + t_{C\downarrow} + 3 \cdot t_{AND\downarrow}) \, . \end{aligned} \qquad \text{(C17)}$$

The constraints derived for $t_{RD\downarrow}$ in cases 1 and 2 (i.e. (C15), (C16) and (C17)) should be considered selecting the minimum delay for the self-resetting loop.

### 2.4 Performance

Here, we derive equations for two important performance factors of the pipeline i.e. *forward latency(L)* and *cycle time(T)*. More importantly, we will show which components of the latter performance metric can be hidden in case of a pipeline with logic processing where the Early Acknowledgement protocol has a competitive edge. We assume that the controller in the middle stage of a pipeline with the same controllers in the previous and next stages. In contrast to the constraint analysis, we assume the controllers are operating at maximal speed in the performance analysis. With these two assumptions the maximum performance of our controller can be derived.

Fig. 9 depicts the Signal Transition Graph(STG) for our controller in desired operation, when it meets the above specified constraints. Thick arrows indicate the signal transitions generated from the environment (previous and next stage controllers) of the controller where as regular arrows indicate transitions made by the controller. Transitions are annotated with the gate delays associated with them. For the delays from the environment, the delays incurred from the controllers of previous and next stages are used. Dashed arrows are for the clock signals of the controller stage and the following stage ($clk_N$ and $clk_{N+1}$) as well as for the data path between the stages, which are not directly in the control path of main control logic, but useful in measuring the cycle time in terms of logic processing delay ($t_{logic}$). For clarity, not all the transition arcs for these two clock signals are shown.

Cycle time is defined as the interval between two successive data items passing through a pipeline stage when the pipeline is operating at maximum speed. We can measure the gate delays between two successive *clk* rising edges for this purpose or equivalently the delay between two successive falling edges of ***Rin***.

First, we will identify the critical cycle of the controller using the STG branch and merge points. The path $Rin- \rightarrow Rout+ \rightarrow Aout+ \rightarrow Rout- \rightarrow Ain+$ is more critical than $Rin- \rightarrow Ain- \rightarrow Rin+ \rightarrow Ain+$ as the delays in the former is larger. Similarly, the path $Rout- \rightarrow Ain+ \rightarrow Rin- \rightarrow Rout+$ is more critical than the path $Rout- \rightarrow Aout- \rightarrow Rout+$. Hence, the critical cycle of the controller lies on the path marked with a thin dashed cycle. In fact, it is required to unfold the STG to previous and next stages as well to formally show that this path with the delays shown in the STG is indeed the critical path defining the cycle time of the controller. The details of the inductive proof which arrives at the same conclusion were left out. The cycle time can be obtained from the critical path as a function of gate delays and required matched delay ($t_{MD}$) as follows.

$$T = 3 \cdot t_{AND\uparrow} + 2 \cdot t_{C\downarrow} + t_{C\uparrow} + t_{MD\uparrow} + t_{MD\downarrow} \, . \qquad (18)$$

In order to obtain cycle time and forward latency in terms of logic processing delay ($t_{logic}$) we need to express the required matched delay $t_{MD}$ for the operations in terms of $t_{logic}$. When the data is latched with $clk_N+$, the next stage clock $clk_{N+1}+$ needs to be made after $t_{flop} + t_{logic}$ delay, where $t_{flop}$ is the delay of the date register. We can relate $t_{logic}$ to $t_{MD}$ by measuring the same delay in two paths to the event of $clk_{N+1}+$.

- Path on control cycle: $Rin- \rightarrow Rout+ \rightarrow Aout+ \rightarrow Rout- \rightarrow clk_{N+1}+$

$$\begin{aligned} T_1 &= t_{AND\uparrow} + t_{C\uparrow} + t_{MD\uparrow} + t_{AND\uparrow} \\ &\quad + t_{C\downarrow} + t_{MD\downarrow} + t_{AND\downarrow} + t_{NOT\uparrow} \, . \end{aligned} \qquad (19)$$

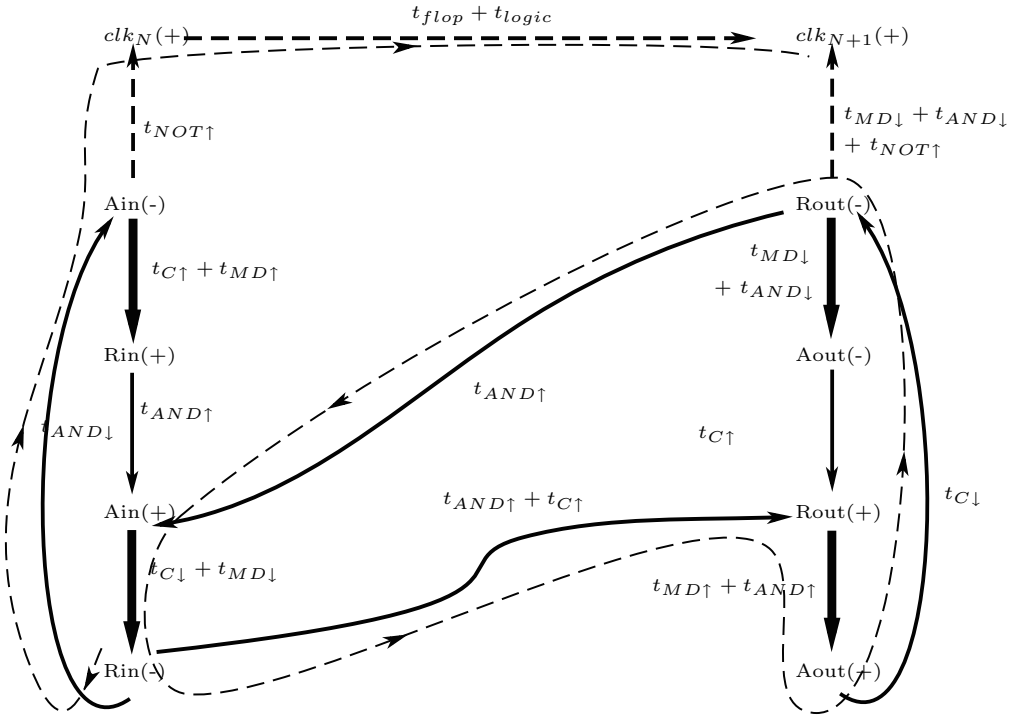


**Fig. 9** STG for Early Acknowledgement Controller.

- Path on data cycle: $Rin- \rightarrow Ain- \rightarrow clk_N+ \rightarrow clk_{N+1}+$

$$T_2 = t_{AND\downarrow} + t_{NOT\uparrow} + t_{flop} + t_{logic} \,. \qquad (20)$$

To ensure the correct operation of the pipeline, $T_1 \geq T_2$ must hold. Thus, from above two equations we can derive an expression for the minimum value of $t_{MD\uparrow}$ as follows:

$$t_{MD\uparrow} \geq (t_{flop} + t_{logic}) - (2 \cdot t_{AND\uparrow} + t_{MD\downarrow} + t_{C\uparrow} + t_{C\downarrow}) \,. \qquad (21)$$

Thus, if

$$t_{logic} \geq (2 \cdot t_{AND\uparrow} + t_{MD\downarrow} + t_{C\uparrow} + t_{C\downarrow}) - t_{flop} \qquad (22)$$

holds, we can find the cycle time in terms of $t_{logic}$ by substituting $t_{MD\uparrow}$ in equation (18) by the right hand side of (21) the cycle time for the linear controller of Early Acknowledgement protocol can be expressed as follows.

$$T^l_{EA} = t_{flop} + t_{logic} + t_{AND\uparrow} + t_{C\downarrow} \,. \qquad (23)$$

Note that in the above expressions $t_{MD\downarrow}$ is equal to $t_{AND\downarrow}$ from our implementation shown in Fig. 3. The convention that we use for cycle time and forward latency consists of the protocol in subscript (*EA, 2P, 4P*, respectively) and the controller type (*l, cb* for linear and conditional branch type controllers) in the superscript.

In the case where logic processing time is smaller and the inequality (22) does not hold, we obtain the minimum cycle time (maximum throughput) of this controller. directly from equation (18) with $t_{MD\uparrow} = t_{MD\downarrow} = 0$, which is:

$$T^l_{EA}|_{min} = 3 \cdot t_{AND\uparrow} + 2 \cdot t_{C\downarrow} + t_{C\uparrow} \,. \qquad (24)$$

The above cycle minimum time is valid since it is possible to remove the matched delay without violating the timing constraints derived for $t_{MD\uparrow}$. This could be confirmed in our expriments as well.

Forward latency is the time taken by a data item to emerge from an initially empty pipeline. Transitions that take place in the forward latency path starting from the $Rin-$ of the STG is shown in the Fig. 9 in a thin dashed line. When the inequality (22) holds, we can have the similar argument to obtain forward latency as follows.

$$L^l_{EA} = t_{AND\downarrow} + t_{NOT\uparrow} + t_{flop} + t_{logic} \,. \qquad (25)$$

When the logic processing delay is small and inequality (22) does not hold, the critical path for forward latency lies on the path: $Rin- \rightarrow Rout+ \rightarrow Aout+ \rightarrow Rout- \rightarrow clk_{N+1}+$, which is:

$$L = t_{AND\uparrow} + t_{C\uparrow} + t_{MD\uparrow} + t_{AND\uparrow} + t_{C\downarrow} + t_{MD\downarrow} + t_{AND\downarrow} + t_{NOT\uparrow} \,. \qquad (26)$$

Similar to the minimum cycle time, we can derive the minimum forward latency on this path with $t_{MD\uparrow} = t_{MD\downarrow} = 0$, which is:

$$L^l_{EA}|_{min} = 2 \cdot t_{AND\uparrow} + t_{AND\downarrow} + t_{C\uparrow} + t_{C\downarrow} + t_{NOT\uparrow} \,. \qquad (27)$$

In a general pipeline with logic processing, the condition (22) often holds. In that case, most of the delays needed for the controller operations are hidden by the logic processing delay, in the cycle time and forward latency as shown in (23) and (25), respectively.

## 3. Comparison to 2-phase and 4-phase Pipeline Controllers

In order to demonstrate the advantage of Early Acknowledgement protocol based controller, we have compared its performance to 2-phase and 4-phase pipeline controllers. The following sections describe the controllers used for this comparison and their key features.

### 3.1 2-phase Controller: The MOUSETRAP

For the 2-phase or the transition signalling protocol, the MOUSETRAP controller is selected for its simplicity and high performance. As shown in Fig. 10, the controller consists of a simple transparent latch and a XONR gate. The same type of latches (instead of D-flipflops) and the latch enable signal $enable$ are used also for the data path.

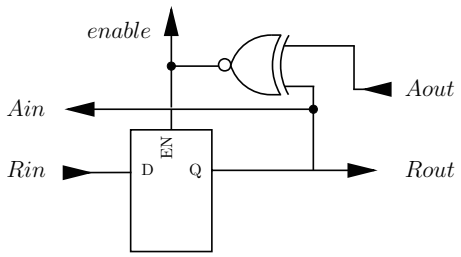


**Fig. 10** MOUSETRAP Controller.

In [4] the operation of the MOUSETRAP controller in both high-speed pipeline and pipelines with logic processing is described in detail. The most important point to note in the view of Early Acknowledgement protocol and 4-phase signalling protocol is that there is no resetting overhead in the 2-phase protocol, hence in the controller as well. The authors have derived the cycle time and the forward latency of the MOUSETRAP controller. The derivation is presented in Appendix A. The result are summaries as follows.

$$T^l_{2P} = 2 \cdot t_{latch} + t_{logic} + t_{XNOR\uparrow} \, . \tag{28}$$

$$L^l_{2P} = t_{latch} + t_{logic} \, . \tag{29}$$

The minimum cycle time and forward latency can be derived from the above equations when $t_{logic} = 0$ as follows.

$$T^l_{2P}|_{min} = 2 \cdot t_{latch} + t_{XNOR\uparrow} \, . \tag{30}$$

$$L^l_{2P}|_{min} = t_{latch} \, . \tag{31}$$

### 3.2 4-phase Controller

We have used the 4-phase controller proposed in [16] for this comparison. The controller is shown in Fig. 11. *G1* and *G2* are complex gates which comprises the controller.

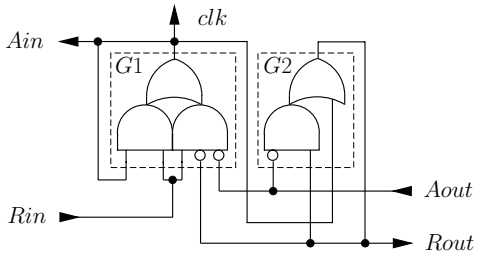


**Fig. 11** 4-phase Controller.

We could derive the cycle time and latency for this 4-phase controller using the similar mechanism employed in controller for Early Acknowledgement protocol. The formal analysis for obtaining the above cycle time and latency is described in Appendix B. The results can be summarized as follows. When $t_{logic}$ is large enough such that the inequality

$$t_{logic} \geq t_{G1\uparrow} + t_{G2\uparrow} - t_{flop} \tag{32}$$

holds, the cycle time and the latency can be expressed as:

$$T^l_{4P} = t_{flop} + t_{logic} + t_{G1\uparrow} + t_{G2\uparrow} + t_{G1\downarrow} + t_{G2\downarrow} + t_{AND\downarrow} \, . \tag{33}$$

$$L^l_{4P} = t_{flop} + t_{logic} + t_{G1\uparrow} \, . \tag{34}$$

When $t_{logic}$ is small that the above inequality does not hold, The cycle time and latency take the following form.

$$T^l_{4P}|_{min} = 2 \cdot (t_{G1\uparrow} + t_{G2\uparrow}) + t_{G1\downarrow} + t_{G2\downarrow} \, . \tag{35}$$

$$L^l_{4P}|_{min} = 2 \cdot t_{G1\uparrow} + t_{G2\uparrow} \, . \tag{36}$$

### 3.3 Comparison of performance

The merits of using Early Acknowledgement controller could be observed in case where $t_{logic}$ satisfies the condition that we derived in (22). Then the cycle time for our controller is given by (23). This can be compared analytically to the 2-phase and 4-phase protocols using equations (28) and (33). It is not possible to compare the cycle times without specific delays from technology libraries. However, we can get an idea of the controller overhead on the overall cycle time in each case. Note that the data-path delay for our controller and 4-phase controller is $t_{flop} + t_{logic}$, since we use D-flipflops on the data-path. In the case of the MOUSETRAP controller data-path delay is $t_{latch} + t_{logic}$ due to the use of transparent latches. Any additional terms appearing in the cycle-time expressions apart from the data-path delays are incurred by the controller overhead. Hence,

our controller has an overhead of only two gate delays ($t_{AND\uparrow} + t_{C\downarrow}$) which is comparable to the 2-phase controller's overhead ($t_{latch} + t_{XNOR\uparrow}$). For 4-phase controller the overhead is 5 gate delays which are incurred by the resetting phase.

However our controller does not exhibit the performance advantage in FIFO types of pipelines where there is no logic processing. In that case its minimum cycle time is given by (24) where controller overhead is exposed in the critical cycle of the controller. Compared to 2-phase controller (30) it is clearly larger though it is in the same order of gate delays (6 gates) in comparison to the 4-phase controller (35).

Forward Latencies of the three controllers can be compared using equations (25), (29) and (34). Evidently, the MOUSETRAP controller exhibits the lowest forward latency where as the 4-phase also controller shows slightly lower latency compared to our controller given the gate delays are equal. The forward latencies when there is no logic processing is given by equations (27), (31) and (36). Again we can observe that the MOUSETRAP controller has the lowest latency and our controller has the highest.

It should be noted that our controller and the MOUSETRAP controller are not Speed Independent(SI) circuits while the 4-phase controller is. The performance advantage of both controllers over 4-phase controller is partly depend on this as well.

## 4. Conditional Branch Controller

We have used the Conditional Branch(CB) non-linear pipeline operation to demonstrate the simplicity of the Early Acknowledgement protocol (which is essentially 4-phase protocol) in composing complex pipeline constructs. First, the abstract operation of the Conditional Branch without any particular reference to a signalling protocol is given followed by the implementation of CB controller for each signalling protocol.

In contrast to Fork operation, Conditional Branch operation diverts the data to only *one* branch depending on *select* signal to the controller. The interface of a two way Conditional Branch controller is shown in Fig. 12.

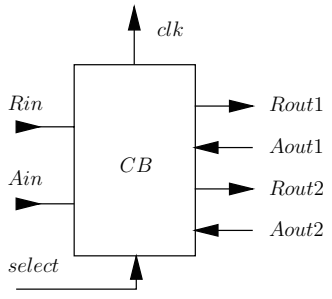


**Fig. 12** Conditional Branch Controller.

Conditional Branch controller communicates with the input stage with *Rin* and *Ain* signals where as the two output stage control signals are *Rout1*, *Aout1* and *Rout2*, *Aout2* respectively. When a request *Rin* is made from the previous stage of pipeline, data is latched by *clk* signal. Acknowledgement *Ain* is sent to the input stage when the data is latched. Depending on *select* signal, request is routed on either the first branch *Rout1* or the second branch *Rout2*. It is assumed that *select* signal is generated from several datapath signals.

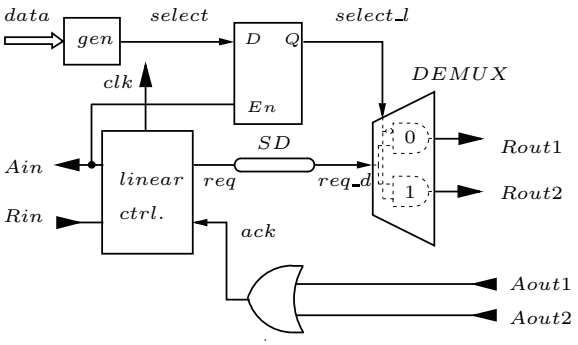


**Fig. 13** Early Acknowledgement CB pipeline controller.

### 4.1 Early Acknowledgement CB Controller

The Conditional Branch controller for Early Acknowledgement protocol is a simple extension of its linear controller. The controller can be composed of a linear controller (for Early Acknowledgement protocol), demultiplexer, transparent latch, delay element ($t_{SD}$) and an OR gate shown in Fig. 13. *Rin* and *Ain* of the controller are handled by the linear controller used within the Conditional Branch controller. *select* signal is latched by a transparent latch using the $Ain$ as latch enable. This ensures the *select* is being sampled in the positive edge of *Ain* and held stable in *select_l* when the request is made on negative edge of *Rin* and the latch is made opaque by *Ain* low. A function generator *gen* which produces *select* from data is explicitly considered for analyzing constraints imposed from such an application. The asymmetric delay element $t_{SD}$, which is the same type as *MD* shown in Fig. 3, is used to compensate for the delay of *gen*. An additional constraint on $t_{SD}$ for this correct sampling of *select* signal is presented in constraint analysis of this controller. The *select_l* diverts the delayed request *req_d* from linear controller to either *Rout1* or *Rout2* conditional paths through the demultiplexer. Since only one request is acknowledged from either *Aout1* or *Aout2*, the acknowledgements from the conditional branches can be simply ORed to produce *ack* to the linear controller.

#### 4.1.1 Timing Constraints

Timing constraints for Conditional Branch controller are analyzed as an extension of linear controller constraints presented in Section 2.3. Again, we obtain timing constraints

to satisfy the desired operation as described in Section 4 assuming the Conditional Branch controller is in a middle stage of a pipeline with environment operating at a speed equal to or slower than our linear controller. First, Constraint 1 and 2 presented in Section 2.3 are reevaluated to ensure the proper operation of the linear controller used within the Conditional Branch controller. Then an additional constraint on $t_{SD}$ for proper operation of demultiplexer is presented.

**Constraint 1 and 2.** The Conditional Branch controller is viewed as a linear controller (or linear pipeline) from the input side, since it employs a linear Early Acknowledgement controller to communicate with *Rin* and *Ain*. Difference can only be perceived when viewed from the branches of the controller. Thus, the constraints involving $t_{Rout\downarrow \rightarrow Aout\downarrow}$ should be reconsidered. This corresponds to **Case 2** of each constraint where $Aout \downarrow$ causes $Rout \uparrow$. The two constraints (C12) and (C17) are restated with an increased delay $t'_{Rout\downarrow \rightarrow Aout\downarrow}$ in output side as follows.

$$t_{MD\uparrow} \geq t'_{Rout\downarrow \rightarrow Aout\downarrow} - (t_{C\downarrow} + t_{AND\uparrow} + 2 \cdot t_{AND\downarrow}) \quad \text{(C37)}$$

$$\text{and,} \quad t_{RD\downarrow} \geq t'_{Rout\downarrow \rightarrow Aout\downarrow} + t_{C\uparrow} - (t_{AND\uparrow} + t_{C\downarrow} + 3 \cdot t_{AND\downarrow}) \; . \quad \text{(C38)}$$

From Fig. 13, we have

$$t'_{Rout\downarrow \rightarrow Aout\downarrow} = 2 \cdot t_{AND\downarrow} + t_{Rout\downarrow \rightarrow Aout\downarrow} + t_{OR\downarrow} \; . \quad \text{(C39)}$$

$t_{MD\uparrow}$ and $t_{RD\downarrow}$ should be selected to satisfy these new constraints.

**Constraint 3.** An additional constraint on Conditional Controller requires *select_l* signal to be valid *before req_d* goes high. This ensures the proper operation of demultiplexer which switches the request *req* to either branch depending on *select* signal. Early Acknowledgement protocol stipulates that *data* becomes valid *before* $Rin \downarrow$ arrives. Thus, the worst case for this constraint is when *data* becomes valid simultaneously with $Rin \downarrow$. In that case, the constraint translates to:

$$t_{Rin\downarrow \rightarrow req\_d\uparrow} \geq t_{Rin\downarrow \rightarrow select\_l} \; . \quad \text{(C40)}$$

In order that $Rin \uparrow$ causes $req\_d \uparrow$, at least $t_{AND\uparrow}$ of *A2*, $t_{C\uparrow}$ of *C*-element and $t_{SD\uparrow}$ of *SD* should occur. Hence the lower bound for the left-hand-side of the above constraint can be expressed as follows.

$$t_{Rin\downarrow \rightarrow req\_d\uparrow} \geq t_{AND\uparrow} + t_{C\uparrow} + t_{SD\uparrow} \; . \quad \text{(C41)}$$

The latch is transparent due to $Ain$ high when $Rin \downarrow$ occurs. Thus, The time for *select_l* to become valid is determined by the delays of *gen* and the latch. That is:

$$t_{Rin\downarrow \rightarrow select\_l} = t_{gen} + t_{latch} \; . \quad \text{(P42)}$$

Hence the constraint on the asymmetric delay can be derived as:

$$t_{AND\uparrow} + t_{C\uparrow} + t_{SD\uparrow} \geq t_{gen} + t_{latch}$$
$$t_{SD\uparrow} \geq t_{gen} + t_{latch} - (t_{AND\uparrow} + t_{C\uparrow}) \; . \quad \text{(C43)}$$

This constraint defines the selection of $t_{SD\uparrow}$ based on the select generator function.

Note that $t_{SD}$ can be "borrowed" from the matched delay to be inserted in *Rout1* and *Rout2* paths. Suppose that the matched delays on the two branches of controller are $t_{MD1\uparrow}$ and $t_{MD2\uparrow}$, and $M = min(t_{MD1\uparrow}, t_{MD2\uparrow})$. Then, $t_{SD\uparrow}$ can be set upto $M$ replacing $t_{MD1\uparrow}$ and $t_{MD2\uparrow}$ by $t_{MD1\uparrow} - M$ and $t_{MD2\uparrow} - M$, respectively. If such $t_{SD\uparrow}$ satisfies all of the above constraints, this Constraint 3 can be satisfied without causing any performance penalty.

### 4.1.2 Performance

In the this section the cycle time and forward latency of the controller are derived analytically as it was done for the linear controller. Minimum time for processing logic, $t_{logic}$ that ensures the advantage of the new controller by hiding its overhead, is also derived in the form of an inequality similar to that of (22).

Fig. 14 shows the STG for the Conditional Branch controller. The arrows with associated delays in square brackets indicate the delays incurred by the extra components (demultiplexer, delay element and OR gate) of the controller. We try to differentiate between linear operation overhead and additional overhead incurred due to the Conditional Branch operation, and then reflect them upon equations that we derive as well. The diagram shows STG for only one branch of the controller (*Rout1/Aout1*) without loosing any functional information necessary to perform the analysis.

The notable point of deviation from the STG of linear controller is in matched delays $t_{MD1}$ and $t_{MD2}$ for the output branches of the controller. As shown in Fig. 13 and detailed in constraint analysis a part of output side matched delays may be used for $t_{SD}$ inside our controller to compensate for the select generator function. The matched delays external to the controller $t'_{MD1}$ and $t'_{MD2}$ are selected such that the original matched delay remains the same. i.e.

$$t_{MD1\uparrow} = t'_{MD1\uparrow} + t_{SD\uparrow}$$
$$\text{and} \quad t_{MD2\uparrow} = t'_{MD2\uparrow} + t_{SD\uparrow} \; .$$

Analogously to the reasoning that we followed for the linear controller, we can measure the delays on control cycle and data cycle to derive the cycle-time in $t_{logic}$ and inequality for optimal operation of the controller. The cycle-time in terms of gate delays as shown in the STG, can be expressed as follows.

$$T = t_{AND\uparrow} + t_{C\uparrow} + [t_{SD\uparrow}] + [t_{AND\uparrow}] + t'_{MD1\uparrow} + t_{AND\uparrow} + [t_{OR\uparrow}] + t_{C\downarrow} + t_{AND\uparrow} + t_{C\downarrow} + t_{MD0\downarrow} \; . \quad \text{(44)}$$

The delays enclosed within square braces indicates the

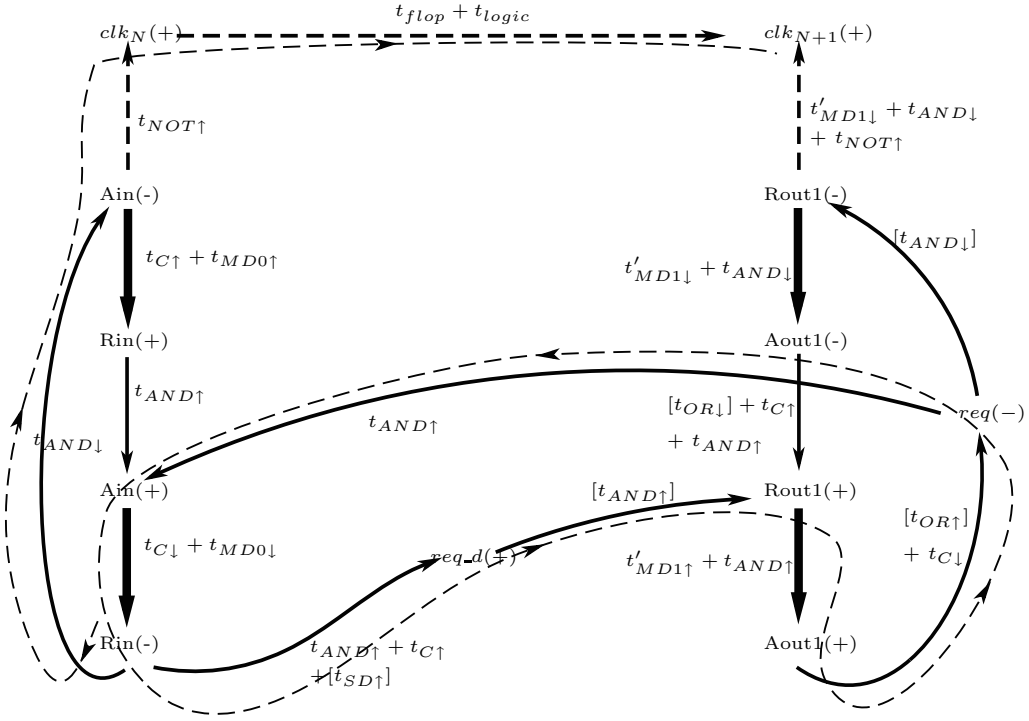


**Fig. 14** STG for Conditional Branch Controller for Early Acknowledgement protocol.

extra delays of the path due to Conditional Branch operation. In order to express the above cycle-time in terms of $t_{logic}$, the delays in the control and data paths can be measured.

- Path on control cycle: $Rin- \rightarrow req\_d+ \rightarrow Rout_1+ \rightarrow Aout_1+ \rightarrow req- \rightarrow Rout_1- \rightarrow clk_{N+1}+$

$$\begin{aligned} T_{CB1} = {} & t_{AND\uparrow} + t_{C\uparrow} + [t_{SD\uparrow}] + [t_{AND\uparrow}] \\ & + t'_{MD1\uparrow} + t_{AND\uparrow} + [t_{OR\uparrow}] + t_{C\downarrow} \\ & + [t_{AND\downarrow}] + t'_{MD1\downarrow} + t_{AND\downarrow} + t_{NOT\uparrow} \,. \end{aligned} \tag{45}$$

- Path on data cycle: $Rin- \rightarrow Ain- \rightarrow clk_N+ \rightarrow clk_{N+1}+$

$$T_{CB2} = t_{AND\downarrow} + t_{NOT\uparrow} + t_{flop} + t_{logic} \,. \tag{46}$$

Again, for proper operation of the pipeline $T_{CB1} \geq T_{CB2}$ must hold, which translates to:

$$\begin{aligned} t'_{MD1\uparrow} \geq {} & (t_{flop} + t_{logic}) \\ & - (2 \cdot t_{AND\uparrow} + t_{C\uparrow} + t_{C\downarrow} + t'_{MD1\downarrow}) \\ & - [t_{SD\uparrow} + t_{AND\uparrow} + t_{AND\downarrow} + t_{OR\uparrow}] \,. \end{aligned} \tag{47}$$

Thus, if

$$\begin{aligned} t_{logic} \geq {} & (2 \cdot t_{AND\uparrow} + t_{C\uparrow} + t_{C\downarrow} + t'_{MD1\downarrow} \\ & + [t_{SD\uparrow} + t_{AND\uparrow} + t_{AND\downarrow} + t_{OR\uparrow}]) - t_{flop} \end{aligned} \tag{48}$$

holds, the cycle time for Conditional Branch controller $T^{cb}_{EA}$ can be expressed in terms of $t_{logic}$ by substituting the minimum of (47) in equation (44) which is:

$$T^{cb}_{EA} = t_{flop} + t_{logic} + t_{AND\uparrow} + t_{C\downarrow} - [t_{AND\downarrow}] \,. \tag{49}$$

We use the fact that $t'_{MD1\downarrow} = t_{MD0\downarrow} = t_{AND\downarrow}$ in accordance with our implementation to simplify the above expression.

According to inequalities of (22) and (48), minimum of $t_{logic}$ required in order to hide the additional overhead incurred by the Conditional Branch Controller is higher than that of linear controller.

In the case where the logic processing time is smaller such that inequality (48) does not hold, we have the minimum cycle time directly from equation (44) with $t'_{MD1\uparrow} = t_{MD0\downarrow} = 0$ which is:

$$\begin{aligned} T^{cb}_{EA}|_{min} = {} & 3 \cdot t_{AND\uparrow} + t_{C\uparrow} + 2 \cdot t_{C\downarrow} \\ & + [t_{SD\uparrow} + t_{AND\uparrow} + t_{OR\uparrow}] \,. \end{aligned} \tag{50}$$

Forward latency is also measured similar to the linear controller and marked in dashed lines on the STG diagram. For sufficiently large $t_{logic}$, forward latency has the same terms (despite the minimum $t_{logic}$ is larger) as in linear controller. i.e.

$$L^{cb}_{EA} = t_{AND\downarrow} + t_{NOT\uparrow} + t_{flop} + t_{logic} \,. \tag{51}$$

When $t_{logic}$ is small and the inequality (48) does not hold,

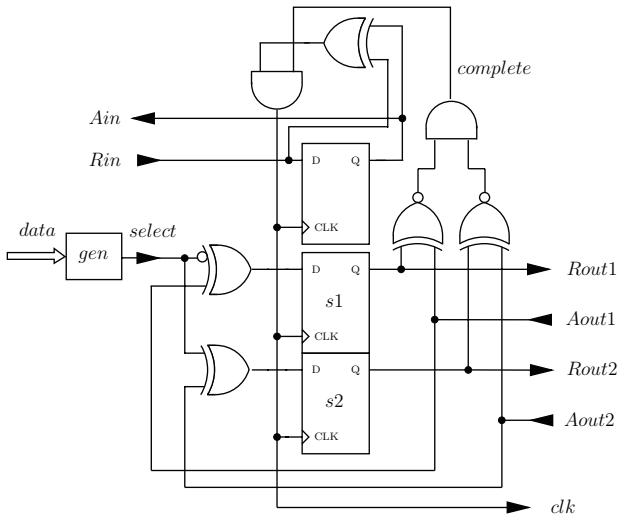


**Fig. 15** 2-phase Conditional Branch Controller.

the critical path lies on the path: $Rin- \rightarrow req\_d+ \rightarrow Rout_1+ \rightarrow Aout_1+ \rightarrow req- \rightarrow Rout_1- \rightarrow clk_{N+1}+$.

$$\begin{aligned} L = & \, t_{AND\uparrow} + t_{C\uparrow} + [t_{SD\uparrow}] + [t_{AND\uparrow}] + t'_{MD1\uparrow} \\ & + t_{AND\uparrow} + [t_{OR\uparrow}] + t_{C\downarrow} + [t_{AND\downarrow}] \\ & + t'_{MD1\downarrow} + + t_{AND\downarrow} + t_{NOT\uparrow} \, . \end{aligned} \tag{52}$$

Similar to the minimum cycle time, the minimum forward latency can be derived for this case when $t'_{MD1\uparrow} = t'_{MD1\downarrow} = 0$ as follows.

$$\begin{aligned} L^{cb}_{EA}|_{min} = & \, 2 \cdot t_{AND\uparrow} + t_{AND\downarrow} + t_{C\uparrow} + t_{C\downarrow} + t_{NOT\uparrow} \\ & + [t_{SD\uparrow} + t_{AND\uparrow} + t_{AND\downarrow} + t_{OR\uparrow}] \, . \end{aligned} \tag{53}$$

### 4.2 2-phase CB Controller

Conditional Branch controller for transition signalling protocol, is not straightforward as in Early Acknowledgement protocol or 4-phase protocol. Since there is no resetting of the request or acknowledgement signal, we cannot make use of a demultiplexer to route the request on the sampled *select* signal. Fig. 15 shows Conditional Branch controller for 2-phase protocol based on [18]. Note that the D-flops are used in contrast to the transparent latches used in the MOUSE-TRAP controller for linear pipeline. The D-flipflop based controller is more robust than the transparent latch based controller in this case, hence we use the former.

Initially, all control signals are at the same state and *complete* signal is high which indicates the operations of the output side of the controller is complete. *select* signal can either be at high or low depending on the data or other control information which handles the branching operation. When a request is made with a transition on *Rin*, difference in states of *Rin* and *Ain* generates *clk* signal which is gated by *complete*. Since *complete* is high initially, the *clk* signal is raised latching the control and data signals. Once *Rin* is latched, the same transition occurs in *Ain* which acknowledges the request to the input side. *s1* and *s2* flipflops work as a "transition demultiplexor" which generates the requests on *Rout1* either *Rout2* depending on *select* signal. Transition on *Rout1* or *Rout2* is made using the previous level of it from *Aout1* or *Aout2* respectively and inverting it through the two XOR gates. The first XOR gate generates "$Rout1 = \overline{Aout1}$ when $select = 0$" where as the second XOR gate generates "$Rout2 = \overline{Aout2}$ when $select = 1$". For example, if *select* signal is low, *s1* latches $\overline{Aout1}$ generating transitions on *Rout1* i.e. requests on first branch.

Either of the request event causes *complete* signal to go low indicating the latched data is being passed to the output stage, which will effectively blocks new requests from the input side. At the acknowledgement of the corresponding branch, each pair of request and acknowledgement signals return to the same state, raising the *complete* signal high and re-enabling the requests from the input side. In comparison to the minimal overhead of linear controller (MOUSE-TRAP), *s1* and *s2* toggle flops to generate requests and completion detection mechanism of the controller incur considerable overhead in the operation, adversely affecting its performance.

#### 4.2.1 Performance

A formal analysis to obtain the cycle time of this controller is presented in Appendix C. It could be observed that,

$$\text{if,} \quad t_{logic} \geq t_{latch} + t_{XNOR\downarrow} \tag{54}$$

holds, the cycle time and forward latency for this controller can be obtained as:

$$\begin{aligned} T^{cb}_{2P} = & \, t_{flop} + t_{logic} + 2 \cdot t_{AND\uparrow} \\ & + (t_{XNOR\uparrow} - t_{XNOR\downarrow}) \, . \end{aligned} \tag{55}$$

$$L^{cb}_{2P} = t_{flop} + t_{logic} + t_{XOR\uparrow} + t_{AND\uparrow} \, . \tag{56}$$

When the above condition does not hold the minimum of these two parameters are obtained as follows.

$$T^{cb}_{2P}|_{min} = t_{flop} + t_{latch} + t_{XNOR\uparrow} + 2 \cdot t_{AND\uparrow} \, . \tag{57}$$

$$L^{cb}_{2P}|_{min} = t_{XOR\uparrow} + t_{AND\uparrow} + t_{flop} + t_{latch} + t_{XNOR\downarrow} \, . \tag{58}$$

### 4.3 4-phase CB Controller

The Conditional Branch for the 4-phase protocol is similar in construction to that of Early Acknowledgement protocol. The construction of the controller is same as in Fig. 13, except for using a 4-phase linear controller in place of the Early Acknowledgement linear controller. The operation as described in previous Section 4.1 is valid for the 4-phase Conditional Branch controller as well.

#### 4.3.1 Performance

The cycle time and forward latency of this controller are obtained in a way similar to the Conditional Branch controller

of Early Acknowledgement protocol. The details are given in Appendix D. The obtained expressions can be summarized as follows.

$$\text{if,} \quad t_{logic} \geq (t_{G1\uparrow} + t_{G2\uparrow} + [t_{SD\uparrow} + t_{AND\uparrow}]) - t_{flop} \tag{59}$$

Then,

$$T^{cb}_{4P} = t_{flop} + t_{logic} + t_{G1\uparrow} + t_{G2\uparrow} + t_{G1\downarrow} + t_{G2\downarrow} + t_{AND\downarrow} + [t_{AND\downarrow} + t_{OR\uparrow} + t_{OR\downarrow}] \, . \tag{60}$$

$$L^{cb}_{4P} = t_{flop} + t_{logic} + t_{G1\uparrow} \, . \tag{61}$$

Otherwise, the minimum of cycle time and forward latency are,

$$T^{cb}_{4P}|_{min} = 2 \cdot (t_{G1\uparrow} + t_{G2\uparrow}) + t_{G1\downarrow} + t_{G2\downarrow} + [t_{SD\uparrow} + t_{AND\uparrow} + t_{AND\downarrow} + t_{OR\uparrow} + t_{OR\downarrow}] \, . \tag{62}$$

$$L^{cb}_{4P}|_{min} = 2 \cdot t_{G1\uparrow} + t_{G2\uparrow} + [t_{SD\uparrow} + t_{AND\uparrow}] \, . \tag{63}$$

### 4.4 Comparison of Performance

Again, an accurate comparison of cycle times requires specific delay values from technology libraries. However we can employ the same mechanism to compare the overhead of the controllers that we employed in the linear controller comparison. In comparison of cycle times, from equations (49), (55) and (60) it can be observed that 4-phase cycle time has high overhead compared to the 2-phase and our controller. Hence, we put more emphasis on comparing the first two controllers as it shows the advantage of our controller over 2-phase protocol. For Early Acknowledgement controller (49) and 2-phase controller (55), an approximate comparison can be done assuming $t_{AND\uparrow} \approx t_{AND\downarrow}$ and $t_{XNOR\uparrow} \approx t_{XNOR\downarrow}$ which further simplifies the cycle times to as follows.

$$T^{cb}_{EA} = t_{flop} + t_{logic} + t_{C\downarrow} \, . \tag{64}$$

$$T^{cb}_{2P} = t_{flop} + t_{logic} + 2 \cdot t_{AND\uparrow} \, . \tag{65}$$

Comparison of the simplified cycle times of controllers (65) and (64) shows that the latter is slightly better provided that,

$$t_{C\downarrow} < 2 \cdot t_{AND\uparrow} \, . \tag{66}$$

This can hold in many technologies mainly owing to the fact that the right-hand-side has the coefficient of 2, giving a slight performance advantage to Early Acknowledgement protocol based controller. In a process technology where gate delays can be chosen to such that $t_{AND\uparrow} < t_{AND\downarrow}$ and $t_{XNOR\uparrow} < t_{XNOR\downarrow}$ (for example using transistor sizing in ASIC technologies) cycle times in *both* cases can be reduced according to the expressions that we obtained ((49) and (55)). Again this give rise to above condition which determines the higher performance of the two controllers. As shown in Section 5.2 in the case of our experiments on FPGA where gate delays are identical, it is observed the Early Acknowledgement protocol cycle time performs slightly better.

When there is no logic processing the cycle times can be compared using equations (50), (57) and (62). With the minimum cycle time, 2-phase controller exhibits the highest performance where as 4-phase controller shows the slowest performance. As in the case of linear controller, this result endorses the 2-phase controller as the best candidate for pipelines with very small or no logic processing in between stages.

The forward latencies of the three Conditional Branch controllers were derived in equations (51), (56) and (61). Given that the gate delays are equal we have roughly the same latencies for our controller and 2-phase controller where as the 4-phase controller has slightly lower latency. The minimum latencies obtained for each type of controller (equations (53), (58) and (63)) when there are no logic processing units inbetween stages also confirms to the same order of latencies with 4-phase being the lowest and our controller being the heighest.

## 5. Implementation and Results

In this section we describe in detail the testcases that we made to evaluate the performance of each controller and the preliminary simulation results.

### 5.1 Implementation

As the proof of concept, we have evaluated the performance of each of the controllers on Xilinx Vertex-4 FPGA. We made maximum efforts to minimize the uncertain path delays in FPGA routing. All control and data path circuits of the designs are placed identically in each case using *rloc* placement constraints of the Xilinx ISE tool. Synthesis options, both general and Xilinx specific ones are tuned to suit asynchronous design synthesis. For example, use of global and regional clock buffers is disabled. Thus, we believe that the results we obtained are comparable with each other with minimum of uncertainty in measurements.

For linear controllers we have created simple 8-bit 4-stage FIFOs, operated by each type of controller. For the Conditional Branch controllers we have built a 8-bit Y-shaped pipelines with 4-stages where 2-stages are in the stem of the pipeline and 2-stages are branched out. The Conditional Branch controller is placed in the second stage of the pipeline. All pipelines were constructed to be 8-bit.

Environment for the pipelines comprised of input generating shift registers and output capturing registers (two registers in the case of Conditional Branch) were operating with minimum overhead which maximizes the performance of the controller under test.

Performance of controllers were evaluated in two cases,

1. pipelines operating without any processing

2. pipelines operating with processing between stages

In the first case, there is minimum delay between stages without any logic processing in-between which evaluates the maximum performance of the controllers for high-speed pipelines. Since there is no logic processing ($t_{logic} = 0$), no matched delays were inserted between stages as well ($t_{MD} = 0$).

In the second case, performance of pipeline controllers for a general scenario of pipelines operating with processing in-between stages is tested. In order to emulate the processing elements we have used simple buffers to delay the data-path. The introduced logic delay was between 6.9ns to 7.2ns (varied depending on the exact routing of the data-path) for each stage of the pipeline. This delay chosen such that it satisfies condition (48) (which in turn satisfies condition (22)) that we derived for Conditional Branch controller for Early Acknowledgement protocol. Thus we could obtain performance of Early Acknowledgement protocol (and other protocol) in the case of a general pipeline with logic processing where these two conditions can be easily satisfied. The matched delays for controllers were tuned starting from a higher delay to the lowest possible where the proper operation of the pipeline is guaranteed.

### 5.2 Results

Post-layout simulation results for Vertex-4 obtained using ModelSim are shown in Table 1.

From the first column of results, it can be observed that the 2-phase controllers outperform the 4-phase and Early Acknowledgement controllers in linear and Conditional Branch operations when there are no processing in-between pipeline stages ($t_{logic} = 0$). Its performance advantage is evident in these cases where a minimum overhead in the controller is desirable. Since $t_{logic} = 0$, the condition (22) for Early Acknowledgement controller does not hold, the overhead of the controller is exposed on the critical cycle time which explains its larger cycle time.

According to the second column of the results table, in the cases where logic processing is present between pipeline stages we observe that the Early Acknowledgement controllers perform better as its overhead got hidden in the required delay between stages. For the Early Acknowledgement controller, the condition (22) holds in this case, the performance is comparable to 2-phase controller in linear operation confirming to the analytical cycle times that we obtained in (23) and (28).

From the last 3 rows of the second column, it can be observed that the Conditional Branch controller for Early Acknowledgement controller outperforms the 4-phase controller and performs slightly better than the 2-phase controller. As we demonstrated in our analysis, the ability of the Early Acknowledgement protocol to hide the control overhead results in the performance gain. In our FPGA implementation, all gates (including the C-elements) are implemented using LUTs which have identical delays simplifying the comparison of cycle times for controllers. Given the equal delays in gates, the difference of cycle times of Conditional Branch controllers derived in (49) and (55) amounts to a one gate delay which is roughly 700ps in the Vertex-4 architecture. Hence, the results are confirming to our formal analysis, subjected to routing delay variations. Even though the gain is relatively low, the simplicity of the Early Acknowledgement protocol as a 4-phase protocol makes it more appealing in this case of non-linear asynchronous pipeline application. As described earlier, when using 2-phase protocol, usually translations from 2-phase to 4-phase is required at some points where level sensitive control is necessary. In such cases, our controller has the added advantage employing a variation of 4-phase protocol and having a performance gain over 2-phase protocol by hiding the additional controller overhead incurred by non-linear operations.

As a measure of area consumption our controller we have measured the resource utilization of FPGA for the our designs. Table 2 shows the resource utilization control path including the matched delays in terms of flipflop and/or latch (denoted as FF/LT) and LUTs separately.

**Table 1** Cycle-times comparison.

| Cycle Time (ns) | without processing | with processing |
|---|---|---|
| Linear | | |
| 2-phase (MOUSETRAP) | 2.6 | 9.9 |
| 4-phase | 3.6 | 12.4 |
| Early Acknowledgement | 4.0 | 10.0 |
| Conditional Branch | | |
| 2-phase | 4.0 | 11.2 |
| 4-phase | 7.1 | 15.1 |
| Early Acknowledgement | 5.6 | 10.6 |

**Table 2** Resource Utilization comparison.

| Resource Utilization | without processing | | with processing | |
|---|---|---|---|---|
| | FF/LT | LUT | FF/LT | LUT |
| Linear | | | | |
| 2-phase | 4 | 4 | 4 | 47 |
| 4-phase | 0 | 8 | 0 | 54 |
| Early Ack. | 0 | 24 | 0 | 51 |
| CB | | | | |
| 2-phase | 8 | 11 | 8 | 86 |
| 4-phase | 2 | 15 | 2 | 81 |
| Early Ack. | 2 | 39 | 2 | 84 |

First two columns shows the resource utilization of pipelines without processing for linear and Conditional Branch controllers of all 3 protocols. Here, 2-phase controllers (both linear and Conditional Branch) exhibit the lowest resource utilization as the linear controller consists of just one latch and XNOR gate even though the Conditional Branch controller is rather complex. Early Acknowledgement controller uses 6 LUTs per linear controller including the logic for self resetting delay, which used most resources.

In last two columns the resource utilization for pipeline with processing for linear and Conditional Branch controllers are shown. For the linear controller we could observe that the resource usage is comparable to that of 2-phase controller and better than 4-phase controller. The reason for this is that Early Acknowledgement controller requires a *smaller* matched delay $t_{MD\uparrow}$ for a given logic processing stage with same $t_{logic}$ compared to other two protocols. Same reasoning goes in the case of Conditional Branch controller, where we could even obtain *lower* resource utilization compared to both 4-phase and 2-phase protocol. Hence we could obtain the performance gains described in earlier with even lower resource utilization for the control path which highlights the advantages of employing Early Acknowledgement protocol.

## 6. Conclusions and Future Work

We have proposed a new pipeline controller for Early Acknowledgement protocol. Its timing constraints were analyzed and performance metrics were derived. When the pipeline has logic processing, the controller can operate with minimal overhead by hiding its overhead in the required matched delay. In such a case, we could obtain cycle-time of controller comparable to 2-phase controller -the MOUSETRAP, both analytically and experimentally.

Furthermore, we could emphasize on the advantages of using Early Acknowledgement protocol which also inherit the simplicity of 4-phase protocol by comparing the Conditional Branch controllers for each protocol. The area usage of the protocol is also comparable to other protocols in the preferred application of this protocol since the required matched delay is smaller requiring less area in the design.

We would like to evaluate and confirm the performance of the controllers on ASIC, like on 65nm technology. Experimental results in such a case is deemed necessary to strengthen our claims of the advantages of using Early Acknowledgement protocol.

### References


[1] C. Mannakkara, T. Yoneda, *Asynchronous Pipeline Controller Based on Early Acknowledgement Protocol*, Proceedings on Applications of Concurrency to System Design, 2008, pages 118-127.

[2] C. Mannakkara, T. Yoneda, *Comparison of Standard Cell based Nonlinear Asynchronous Pipelines*, IEICE Technical Report, VLSI, 2007, pages 49-54.

[3] N.Sretasereekul, et. al., *A Zero-Time-Overhead Asynchronous Four-Phase Controller*, Proc. of IEEE International Symposium on Circuits and Systems, 2003, pages V-205 - V-208.

[4] M. Singh, S.M. Nowick, *MOUSETRAP: Ultra-High-Speed Transition-Signaling Asynchronous Pipelines*, Proceedings on Computer Design, 2001, pages 9-17.

[5] I. E. Sutherland *Micropipelines*, Communications of the ACM, 1989, pages 720-738.

[6] S. B. Furber, P. Day, *Four-phase micropipeline latch control circuits*, IEEE Transactions on VLSI Systems, 1996, pages 247-253.

[7] E. Brunvand, *Using FPGAs to Implement Self-Timed Systems*, Journal of VLSI Signal Processing, 1993, pages 173-190.

[8] E. Brunvand, *Translating Concurrent Communicating Programs into Asynchronous Circuits*, Ph.D. Dissertation, Carnegie Mellon University, 1991.

[9] T. Yoneda, et. al. *High Level Synthesis of Timed Asynchronous Circuits*, Proceedings on Asynchronous Circuits and Systems, 2005, pages 178-189.

[10] R.O. Ozdag, et. al. *High-Speed Non-Linear Asynchronous Pipelines*, Proceedings on Design, Automation and Test in Europe, 2002, pages 1000-1007.

[11] Quoc Thai Ho, et. al. *Implementing Asynchronous Circuits on LUT Based FPGAs*, Proceedings on The Reconfigurable Computing Is Going Mainstream, 2002, pages 36-46.

[12] Y. Sato, et. al. *Systematic Reducing of Metastable Operation Occurred in CMOS D Flip-Flops* Systems and Computers in Japan, 2000.

[13] A. Peeters, *Support for Interface Design in Tangram* Asynchronous Interfaces: Tools, Techniques, and Implementations, 2000, pages 57-64.

[14] K.V. Berkel, F. Huberts, A. Peeters, *Streching Quasi Delay Insensitivity by Means of Extended Isochronic Forks* Proceedings on Asynchronous Design Methodologies, 1995, pages 99-107.

[15] M. Singh, S.M. Nowick, *High-Throughput Asynchronous Pipelines for Fine-Grain Dynamic Datapaths*, Proceedings on Advanced Research in Asynchronous Circuits and Systems, 2000, pages 198-209.

[16] I. Blunno, et. al. *Handshake protocols for de-synchronization*, International Symposium on Asynchronous Circuits and Systems, 2004, pages 149-158.

[17] M. Ampalam, M. Singh *Counterflow Pipelining: Architectural Support for Preemption in Asynchronous Systems using Anti-Tokens* Proceedings on International Conference on Computer-aided Design, 2006, pages 611-618.

[18] Private communication with Montek Singh


**Chammika Mannakkara** received BSc. in Electrical and Electronic Engineering from the Faculty of Engineering, University of Peradeniya, Sri Lanka in 2000. He joined Royal Institute of Technology, Sweden where he completed his MSc. in System-on-Chip Design in 2006. Mr. Mannakkara is currently a PhD. candidate at National Institute of Informatics, Tokyo mentored by Dr. Yoneda.

**Tomohiro Yoneda** received B.E., M.E., and Dr. Eng. degrees in Computer Science from the Tokyo Institute of Technology, Tokyo, Japan in 1980, 1982, and 1985, respectively. In 1985 he joined the staff of Tokyo Institute of Technology, and he moved to National Institute of Informatics in 2002, where he is currently a Professor. He was a visiting researcher of Carnegie Mellon University from 1990 to 1991. His research activities currently focus on formal verification of hardware. Dr. Yoneda is a member of IEEE, Institute of Electronics, Information, and Communication Engineers of Japan, and Information Processing Society of Japan.

## Appendix A: Performance analysis of 2-phase linear controller

Here we use the STG presented in [4] (with our naming conventions for the control signals) to derive the performance of the MOUSETRAP controller. Also note that the signals request/acknowledge signals confirm to 2-phase protocol hence an arrow to/from those signals imply a "transition". The cycle time lies on the path marked in dashed line and the forward latency can be measured on the same path from $Ain_N \rightarrow Rin_{N+1} \rightarrow Aout_N$. Hence,

$$\begin{aligned} T &= t_{MD} + t_{latch} + t_{XNOR\uparrow} + t_{latch} \\ &= t_{MD} + 2 \cdot t_{latch} + t_{XNOR\uparrow} \,. \qquad \text{(A·1)} \\ L &= t_{MD} + t_{latch} \,. \qquad \text{(A·2)} \end{aligned}$$

To express the above equations in terms of $t_{logic}$, the time can be measured on control path and data path as follows.

- Path on control cycle: $Ain_N \rightarrow Rin_{N+1} \rightarrow Aout_N \rightarrow En_{N+1}-$

$$T_1 = t_{MD} + t_{latch} + t_{XNOR\downarrow} \,. \qquad \text{(A·3)}$$

- Path on data cycle: $Ain_N \rightarrow En_N- \rightarrow En_{N+1}-$

$$T_2 = t_{XNOR\downarrow} + t_{logic} + t_{latch} \,. \qquad \text{(A·4)}$$

To correctly latch the data at the next stage ($En_{N+1}-$) it is required that $T_1 \geq T_2$ which lead to below condition:

$$\begin{aligned} t_{MD} + t_{latch} + t_{XNOR\downarrow} &\geq t_{XNOR\downarrow} + t_{logic} + t_{latch} \\ t_{MD} &\geq t_{logic} \,. \qquad \text{(A·5)} \end{aligned}$$

In other words, the matched delay should be selected to be equal or greater than the logic delay. The optimal matched delay is when $t_{MD} = t_{logic}$. The cycle time and forward latency will be:

$$\begin{aligned} T &= t_{logic} + 2 \cdot t_{latch} + t_{XNOR\uparrow} \,. \qquad \text{(A·6)} \\ L &= t_{logic} + t_{latch} \,. \qquad \text{(A·7)} \end{aligned}$$

In contrast to Early Acknowledgement controllers, the above equations holds for *any* logic delay making it a very high performance pipeline controller specially when the logic processing is very low limited to one or two gate delays. The maximum performance (minimum cycle time and latency) for this controller is when $t_{logic} = 0$ which can be given as:

$$\begin{aligned} T|_{min} &= 2 \cdot t_{latch} + t_{XNOR\uparrow} \,. \qquad \text{(A·8)} \\ L|_{min} &= t_{latch} \,. \qquad \text{(A·9)} \end{aligned}$$

## Appendix B: Performance analysis of 4-phase linear controller

The STG for obtaining the cycle time and latency is shown in Fig. A·2. Quite evidently the cycle time of the 4-phase controller has controller overhead which lies on the critical cycle and it cannot be hidden by the matched delay, unlike Early Acknowledgement controller. The analysis of the cycle time and latency is similar to the Early Acknowledgement controller presented in 2.4, hence we left out trivial deductions that can be made directly from the STG diagram. We used the $Rin+$ as the starting transition confirming to the semantics of the 4-phase protocol. The critical path lies on the dashed line path which constitutes a twisted loop. Hence, the cycle time and forward latency can be obtained as a function of gate delays (starting from $Rin+$) as follows.

$$\begin{aligned} T &= t_{G1\uparrow} + t_{G2\uparrow} + t_{MD\uparrow} + t_{G1\uparrow} \\ &\quad + t_{G2\downarrow} + t_{MD\downarrow} + t_{G1\downarrow} + t_{G2\uparrow} \,. \qquad \text{(A·10)} \\ L &= t_{G1\uparrow} + t_{G2\uparrow} + t_{MD\uparrow} + t_{G1\uparrow} \,. \qquad \text{(A·11)} \end{aligned}$$

To bring in $t_{logic}$ to the above equations two paths on control and data cycles are considered.

- Path on control cycle: $Rin+ \rightarrow Ain+ \rightarrow Rout+ \rightarrow Aout+$

$$T_1 = t_{G1\uparrow} + t_{G2\uparrow} + t_{MD\uparrow} + t_{G1\uparrow} \,. \qquad \text{(A·12)}$$

- Path on data cycle: $Rin+ \rightarrow Ain+ \rightarrow Aout+$

$$T_2 = t_{G1\uparrow} + t_{flop} + t_{logic} \,. \qquad \text{(A·13)}$$

From the $T_1 \geq T_2$ condition for proper operation we have:

$$\begin{aligned} t_{MD\uparrow} &\geq t_{flop} + t_{logic} - (t_{G1\uparrow} + t_{G2\uparrow}) \,. \\ \text{Thus, if,} \quad t_{logic} &\geq t_{G1\uparrow} + t_{G2\uparrow} - t_{flop} \qquad \text{(A·14)} \end{aligned}$$

holds, the cycle time and forward latency of the 4-phase controller can be expressed as follows.

$$\begin{aligned} T &= t_{flop} + t_{logic} + t_{G1\uparrow} + t_{G2\uparrow} \\ &\quad + t_{G1\downarrow} + t_{G2\downarrow} + t_{AND\downarrow} \,. \qquad \text{(A·15)} \\ L &= t_{flop} + t_{logic} + t_{G1\uparrow} \,. \qquad \text{(A·16)} \end{aligned}$$

When the above condition does not hold the above parameters can be deduced from equations (A·10) and (A·11) at $t_{MD\uparrow} = t_{MD\downarrow} = 0$.

$$\begin{aligned} T|_{min} &= t_{flop} + t_{logic} + t_{G1\uparrow} + t_{G2\uparrow} \\ &\quad + t_{G1\downarrow} + t_{G2\downarrow} + t_{AND\downarrow} \,. \qquad \text{(A·17)} \\ L|_{min} &= t_{flop} + t_{logic} + t_{G1\uparrow} \,. \qquad \text{(A·18)} \end{aligned}$$

## Appendix C: Performance analysis of Conditional Branch controller for 2-phase protocol

The STG for the functional operation of the 2-phase Conditional Branch Controller is given in Fig. A·3. In this STG also, transitions for only one branch of the controller is given. Note that for signals $Rin$, $Ain$, $Rout_1$ and $Aout_1$ STG implies a "transition" without explicitly unfolding the particular transitions (from low-to-high and high-to-low)

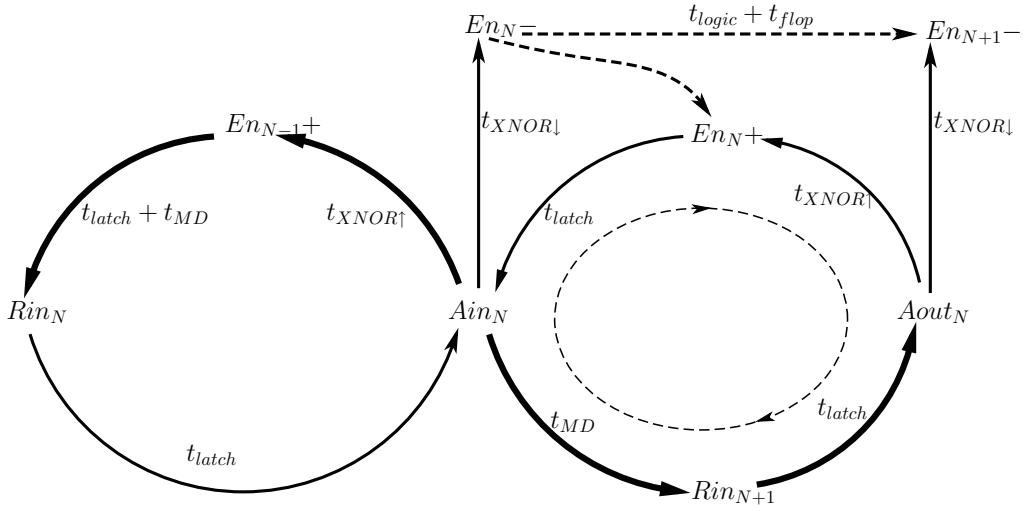


**Fig. A· 1** STG for the MOUSETRAP Controller.

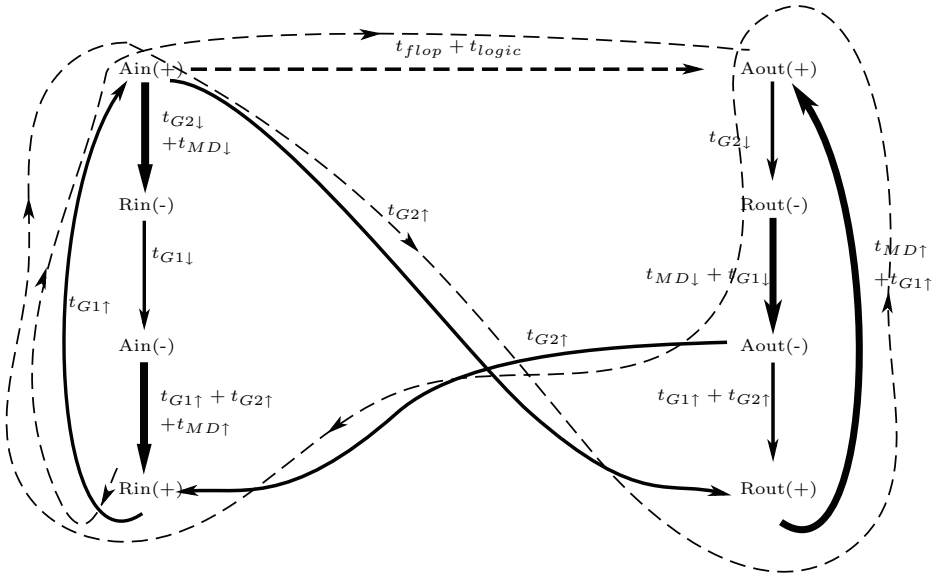


**Fig. A· 2** STG for Four phase Controller.

which are identical in the protocol. The matched delays are symmetrical for the transition signalling protocol. Hence there is no distinction made between $t_{MD\uparrow}$ and $t_{MD\downarrow}$ as in the case of other two protocols.

We can calculate the cycle time and forward latency in terms of $t_{logic}$ using the same rationale used in Conditional Branch controller of the Early Acknowledgement protocol. The cycle time and forward latency of the controller as shown in the STG with thin dashed lines, can be expressed as follows.

$$T = t_{flop} + t_{MD} + t_{latch} + t_{XNOR\uparrow} + 2 \cdot t_{AND\uparrow} \,. \quad \text{(A· 19)}$$

$$L = t_{XOR\uparrow} + t_{AND\uparrow} + t_{flop} + t_{MD} + t_{latch} + t_{XNOR\downarrow} \,. \quad \text{(A· 20)}$$

In order to express above in terms of $t_{logic}$ we measure the delays on the control and data paths.

- Path on control cycle: $clk_N+ \rightarrow Rout_1 \rightarrow Aout_1 \rightarrow En-$

$$T_{CB1} = t_{flop} + t_{MD} + t_{latch} + t_{XNOR\downarrow} \,. \quad \text{(A· 21)}$$

- Path on data cycle: $clk_N+ \rightarrow En-$

$$T_{CB2} = t_{flop} + t_{logic} \,. \quad \text{(A· 22)}$$

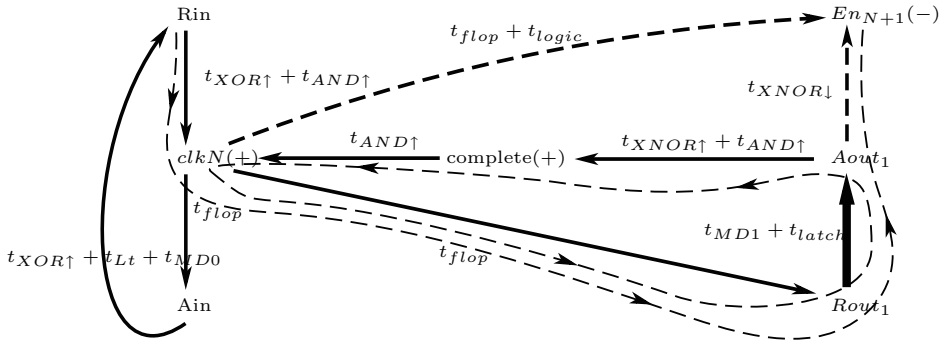


**Fig. A· 3** STG for Conditional Branch Controller for 2-phase protocol.

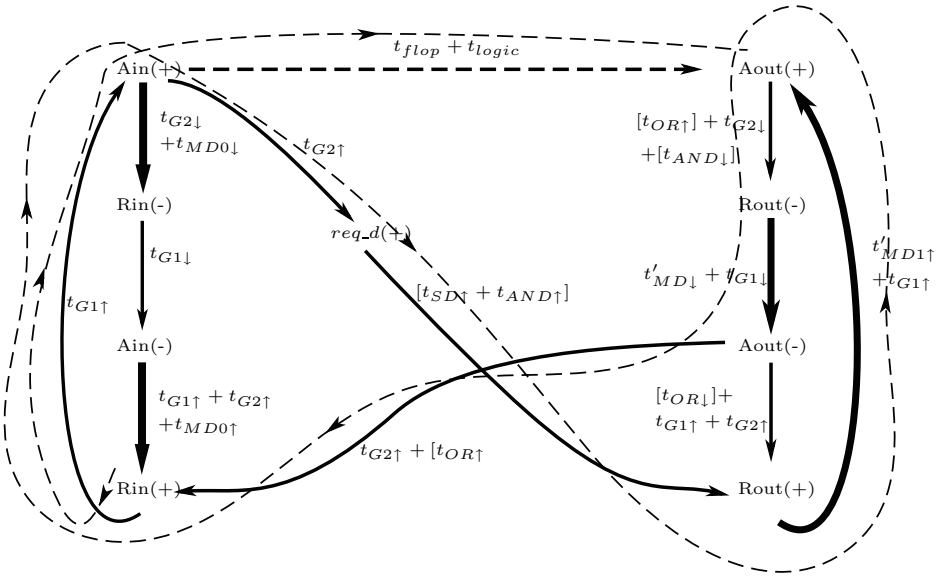


**Fig. A· 4** STG for Conditional Branch Controller for Four phase protocol.

Since $T_{CB1} \geq T_{CB2}$ for proper operation, we can obtain the constraint on $t_{MD1}$ as:

$$t_{MD} \geq t_{logic} - (t_{latch} + t_{XNOR\downarrow}) \; . \quad (A\cdot 23)$$

$$\text{Thus, if,} \quad t_{logic} \geq t_{latch} + t_{XNOR\downarrow} \quad (A\cdot 24)$$

holds, the cycle time and latency for this controller can be obtained as:

$$T = t_{flop} + t_{logic} + 2 \cdot t_{AND\uparrow} + (t_{XNOR\uparrow} - t_{XNOR\downarrow}) \; . \quad (A\cdot 25)$$

$$L = t_{flop} + t_{logic} + t_{XOR\uparrow} + t_{AND\uparrow} \; . \quad (A\cdot 26)$$

When the above condition does not hold we can derive the cycle time and latency directly from (A· 19) and (A· 20) at $t_{MD} = 0$ as follows.

$$T|_{min} = t_{flop} + t_{latch} + t_{XNOR\uparrow} + 2 \cdot t_{AND\uparrow} \; . \quad (A\cdot 27)$$

$$L|_{min} = t_{XOR\uparrow} + t_{AND\uparrow} + t_{flop} + t_{latch} + t_{XNOR\downarrow} \; . \quad (A\cdot 28)$$

## Appendix D: Performance analysis of Conditional Branch controller for 4-phase protocol

Fig. A· 4 shows the STG for the Conditional Branch controller for 4-phase protocol. The analysis of the cycle time and forward latency is similar to that of Early Acknowledgement protocol Conditional Branch controller presented in section 4.1.2.

The arrows with associated delays in square brackets indicate the delays incurred by the extra components (demultiplexer, delay element and OR gate) of the controller. With the same reasoning that we followed for Conditional Branch controllers for Early Acknowledgement protocol, we can derive expressions for cycle time and forward latency in terms of $t_{logic}$. The cycle time and forward latency as marked in thin dashed lines in the STG, can be expressed in terms of gate delays as follows.

$$T = t_{G1\uparrow} + t_{G2\uparrow} + [t_{SD\uparrow} + t_{AND\uparrow}] + t'_{MD1\uparrow} + t_{G1\uparrow} + [t_{OR\uparrow}] + t_{G2\downarrow} + [t_{AND\downarrow}] + t'_{MD1\downarrow} + t_{G1\downarrow} + [t_{OR\downarrow}] + t_{G2\uparrow} \; . \quad \text{(A·29)}$$

$$L = t_{G1\uparrow} + t_{G2\uparrow} + [t_{SD\uparrow} + t_{AND\uparrow}] + t'_{MD1\uparrow} + t_{G1\uparrow} \; . \quad \text{(A·30)}$$

In order to express above two parameters in terms of $t_{logic}$ two paths on control and data cycles are considered.

- Path on control cycle: $Rin+ \rightarrow Ain+ \rightarrow req\_d+ \rightarrow Rout+ \rightarrow Aout+$

$$T_1 = t_{G1\uparrow} + t_{G2\uparrow} + [t_{SD\uparrow} + t_{AND\uparrow}] + t'_{MD\uparrow} + t_{G1\uparrow} \; . \quad \text{(A·31)}$$

- Path on data cycle: $Rin+ \rightarrow Ain+ \rightarrow Aout+$

$$T_2 = t_{G1\uparrow} + t_{flop} + t_{logic} \; . \quad \text{(A·32)}$$

From the $T_1 \geq T_2$ condition for proper operation we have:

$$t'_{MD\uparrow} \geq t_{flop} + t_{logic} - (t_{G1\uparrow} + t_{G2\uparrow} \; . + [t_{SD\uparrow} + t_{AND\uparrow}])$$

$$\text{Thus, if,} \quad t_{logic} \geq (t_{G1\uparrow} + t_{G2\uparrow} + [t_{SD\uparrow} + t_{AND\uparrow}]) - t_{flop} \quad \text{(A·33)}$$

holds, the cycle time and forward latency of the 4-phase controller can be expressed as follows.

$$T = t_{flop} + t_{logic} + t_{G1\uparrow} + t_{G2\uparrow} + t_{G1\downarrow} + t_{G2\downarrow} + t_{AND\downarrow} + [t_{AND\downarrow} + t_{OR\uparrow} + t_{OR\downarrow}] \; . \quad \text{(A·34)}$$

$$L = t_{flop} + t_{logic} + t_{G1\uparrow} \; . \quad \text{(A·35)}$$

When the above condition does not hold the minimum values for above parameters can be deduced from equations (A·29) and (A·30) at $t'_{MD\uparrow} = t'_{MD\downarrow} = 0$.

$$T|_{min} = 2 \cdot (t_{G1\uparrow} + t_{G2\uparrow}) + t_{G1\downarrow} + t_{G2\downarrow} + [t_{SD\uparrow} + t_{AND\uparrow} + t_{AND\downarrow} + t_{OR\uparrow} + t_{OR\downarrow}] \; . \quad \text{(A·36)}$$

$$L|_{min} = 2 \cdot t_{G1\uparrow} + t_{G2\uparrow} + [t_{SD\uparrow} + t_{AND\uparrow}] \; . \quad \text{(A·37)}$$