ISSN 1346-5597

NII Technical Report

Asynchronous Pipeline Controller Based on Early Acknowledgement Protocol

Chammika Mannakkara, and Tomohiro Yoneda

NII-2009-015E
Sept. 2009

PAPER

Asynchronous Pipeline Controller Based on Early Acknowledgement Protocol

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

SUMMARY A new pipeline controller based on the Early Acknowledgement (EA) protocol is proposed for bundled-data asynchronous circuits. The EA 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 the resetting period 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 then discuss their appropriateness. The performance of the controller is compared both analytically and experimentally with those of two other pipeline controllers, namely, a very high-speed 2-phase controller and an ordinary 4-phase controller. Our controller performs better than 4-phase controller when pipeline has processing elements.

We have obtained interesting results in the case of a non-linear pipeline with a Conditional Branch (CB) operation. Our controller has slightly better performance even compared to 2-phase controller in the case of a pipeline with processing elements. Its superiority lies in the EA protocol, which employs return-to-zero control signals like the 4-phase protocol. Hence, our controller for CB operation is simple in construction just like the 4-phase controller. A 2-phase controller for the same operation needs to have a slightly complicated mechanism to handle the 2-phase operation because of the non-return-to-zero control signals, and this results in a performance overhead.

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

1. Introduction

Digital design embraced the synchronous design methodology because it greatly simplifies the design task with a single orchestrator– the clock signal– in command. In the days when CMOS technology was in its infancy, with low gate densities and low operating frequencies, the asynchronous design methodology gave way to a rapid advancement in digital design based on synchronous circuits. However, with the remarkable conformance to Moore's Law to date, the designs have become denser and operating frequencies have increased by several orders of magnitude. Clock skew has become a significant portion of the cycle, and routing the clock signal of a design has become an engineering art. Moreover, clock distribution demands a significant chip area to reduce the skew increasing the power consumption.

The distinctive advantages of asynchronous design, which counter the above problems, signify its importance. Without a global clock signal, the designs eliminate the ever increasing problems associated with it. Each component operates only on request in an inherently power-efficient manner, generating little electromagnetic noise. Component timing is naturally elastic with each component operating at its own speed. This gives a system average-case performance as opposed to worst-case performance dictated by the slowest component in a synchronous system. It also makes the system more resilient to fabrication and environmental variations. Moreover, metastability issues arising from the handling of inputs and/or multiple clock domains do not exist in the asynchronous design paradigm.

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

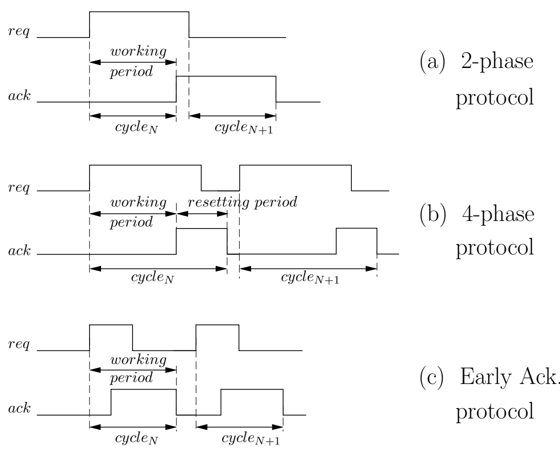


Fig. 1 Handshake protocols.

Many asynchronous pipeline controllers have been proposed over the years [4, 6, 7, 11, 16, 18]. They mainly use either the 2-phase signalling protocol or the 4-phase signalling protocol.

In the 2-phase, or transition signalling, protocol, each cycle consists of two transitions (rising or falling edge) on request (*req*) and acknowledge (*ack*) signals as shown in Fig. 1(a). A request is made by a transition on the *req* signal. The receiver also acknowledges completion of work by a transition on the *ack* signal. The working period is defined as the duration from the request being made to the acknowledgment of completion. Hence, for the 2-phase protocol it is between any transition on the *req* signal to the same transition on the *ack* signal. It should be noted that in this protocol, the next cycle can be started immediately after the working period. The MOUSETRAP [4], a simple and robust linear pipeline controller, is based on this protocol, which has been proven to operate high-throughput pipelines at 2.1–2.4 GHz [5]. However, when the transition signalling protocol is used, translations from 2-phase to 4-phase are usually required at some points, because in many cases, environment circuits use level-sensitive controls.

The 4-phase protocol is shown in Fig. 1(b). As the name suggests, in this protocol, there are four transitions on the *req* and *ack* signals per cycle. Here, a request is made by the rising edge of the *req* signal. The corresponding acknowledgement is indicated by the receiver raising the *ack* signal. The working period for this protocol is from the rising edge of *req* to the rising edge of *ack*. At the end of the working period, both *req* and *ack* signals are high. Upon receipt of the acknowledgement, *req* is lowered and *ack* is also lowered, completing the cycle. The duration in which both *req* and *ack* return-to-zero before the next cycle starts is the resetting period. The different sequencing of these 4-phase signalling transitions leads to different controllers for a range of cost and performance options as shown in [7].

The pipeline controller presented in this paper employs the Early Acknowledgement (EA) protocol introduced in [10], where its original idea was presented in [3]. This protocol is an improvement over the simple 4-phase protocol and can hide the resetting period of the signalling. As shown in Fig. 1(c), the request is made at the rising edge of the *req* signal like in the 4-phase protocol. However, the *ack* signal goes high at any time point after the request, and the *req* signal can be lowered in response to this *early acknowledgement*. In the EA protocol in particular, the completion of work is indicated by the *falling edge* of the *ack* signal in contrast to the 4-phase protocol where it is indicated by the rising edge. Since the working period is from the request being made to notification of completion, in the case of EA protocol it is from the rising edge of the *req* signal to the falling edge of the *ack* signal. Unlike in the 4-phase protocol, at the end of the working period, both signals are reset back to zero and the next cycle can be started immediately. Hence, this protocol eliminates the resetting period 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 the EA protocol. It is an improvement of the controller that we proposed earlier in [2]. We also describe how this controller can be used for non-linear conditional branch (CB) operation. For both cases, we show the set of timing constraints to be satisfied for proper operation of the controller. Finally, this paper shows analytical and experimental performance comparisons with the existing 2- and 4-phase controllers.

The rest of the paper is organized as follows. Section 2 describes 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 with the 2- and 4-phase linear controllers is given in Section 3. The design and analysis of the CB non-linear controller is given in Section 4. Section 5 presents comparative experimental results for the three controllers. Section 6 presents conclusions and mentions future work in this research.

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

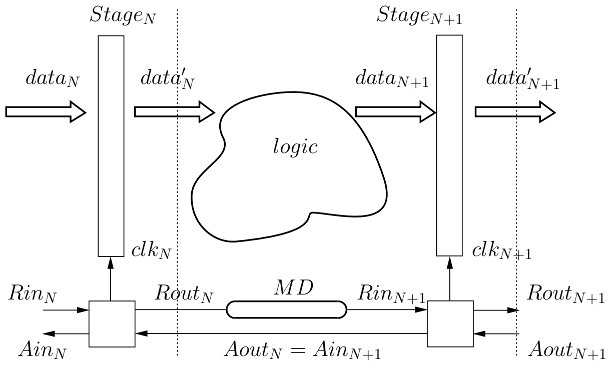


Fig. 2 General pipeline with logic processing.

2. Pipeline Controller for EA Protocol

2.1 Pipeline Operation of EA Protocol

First, we define a few naming conventions that we use for all controllers throughout the rest of this paper. A general diagram of a pipeline using the 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 to the controller of $stage_N$, and Ain_N is the corresponding acknowledgement signal from it to the input side ($stage_{N-1}$). Similarly, $Rout_N$ and $Aout_N$ are the request and acknowledgement to and from the output side ($stage_{N+1}$) controller. The local clock signal of the stage generated by the controller is clk_N. The logic processing unit ($logic$) between the pipeline stages is accounted for by the matched delay (MD) inserted in the request line between stages. For the 2-phase protocol, the delay can be symmetric such as a string of buffers, whereas for the 4-phase protocol (hence, for the EA 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. In our implementation, $t_{MD\downarrow}$ is equal to $t_{AND\downarrow}$.

Fig. 4 shows the operational waveforms of the EA controller in a general pipeline with logic processing as in Fig. 2. The EA protocol uses the falling edge of the acknowledgement signal to indicate the completion of the working period. Hence, data D on $data_N$ will be captured in $stage_N$ at the falling edge of Ain_N (i.e., $clk_N = \overline{Ain_N}$). The captured data is processed by the $logic$ unit in between two

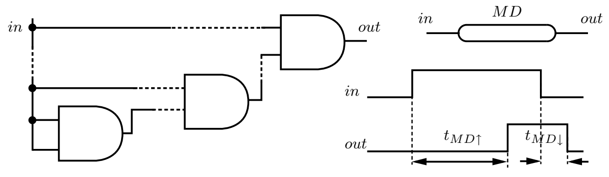


Fig. 3 Asymmetric delay for MD.

stages and becomes available at $data_{N+1}$ after t_{logic} delay. Concurrently, on the control path, $Rout_N$ is raised and $stage_{N+1}$ controller receives the request on Rin_{N+1} after the $t_{MD\uparrow}$ delay. The matched delay MD is chosen such that the falling edge of Ain_{N+1} occurs just after $data_{N+1}$ becomes valid. This essentially hides the controller overhead (i.e., from the rising edge of Rin_{N+1}, up to the falling edge of Ain_{N+1}) inside the required t_{logic} delay by offsetting it from the matched delay MD. Thus,

$$t_{logic} = t_{MD} + t_{ctrl} \tag{1}$$

where t_{ctrl} is the processing delay of the controller. When $t_{logic} \geq t_{ctrl}$, it can be completely hidden inside. However, for fine-grained pipelines with 1–2 gates per stage, this condition may not hold, and in that case, the controller delay is exposed to the pipeline operation. Hence, our controller is preferable in applications where there is a fairly large processing delay (t_{logic}) between stages.

2.2 Controller Operation

Our controller for the EA protocol is depicted in Fig. 5. The controller consists of two AND gates, a C-element, an inverter, and an asymmetric delay (RD) for the self-resetting of the *complete* signal. The implementation of this delay is shown in Fig. 6. $t_{RD\downarrow}$ is the variable part of the delay, and $t_{RD\uparrow}$ equals to $t_{OR\uparrow}$. The clock signal *clk* of the pipeline stage is derived from $\overline{Ain}$, and allows the clocking of the stage to be made at the falling edge of the acknowledgement.

Fig. 7 shows the operation of the controller, which conforms to the pipelined operation in Fig. 4. Initially, all the control signals are low except for the *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 because there are no pending requests at the output stage through *Rout* (For the blocked case, see below).

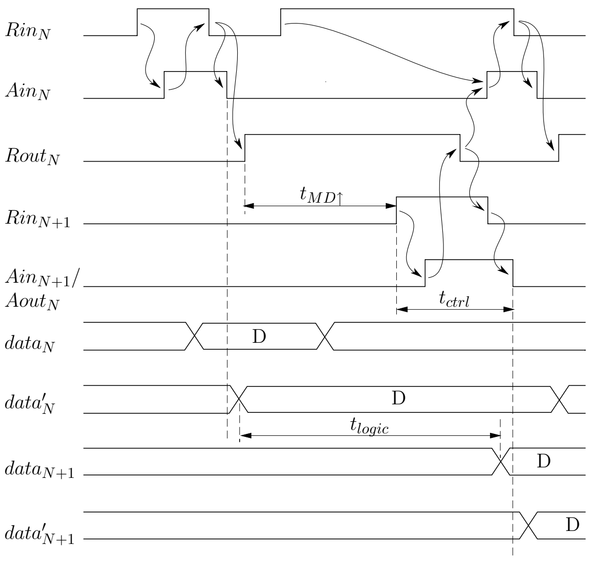


Fig. 4 Behavior of EA controller.

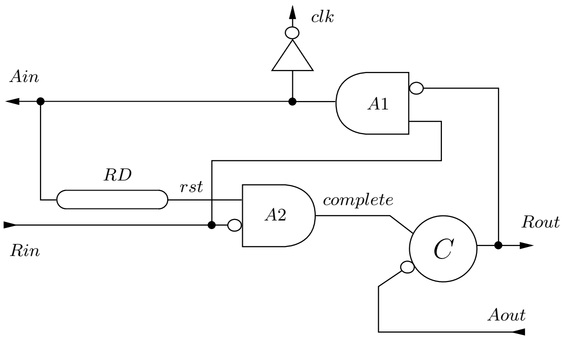


Fig. 5 EA pipeline controller.

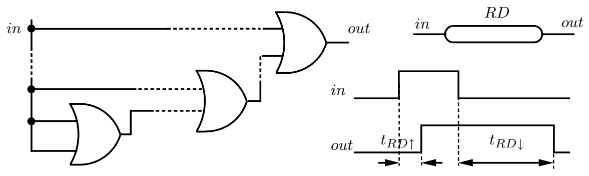


Fig. 6 Asymmetric Delay for RD.

Since the early acknowledgement is provided by raising *Ain*, *rst*– the input for the *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 edge 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 the *Rout* has been driven high, it can be maintained high by C-element[†] even after the *complete* signal has been 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 to correctly produce the $Rout$ signal.

Since the controller has fully completed the handshake cycle at the input side, it is free to make a new request on *Rin*. However, the pending output request *Rout* high effectively blocks the generation of an early acknowledgement back to the input side. Upon receipt of acknowledgement high on *Aout*, *Rout* will be lowered, and the blocked request from the input stage will be free to send the early acknowledgement by 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.

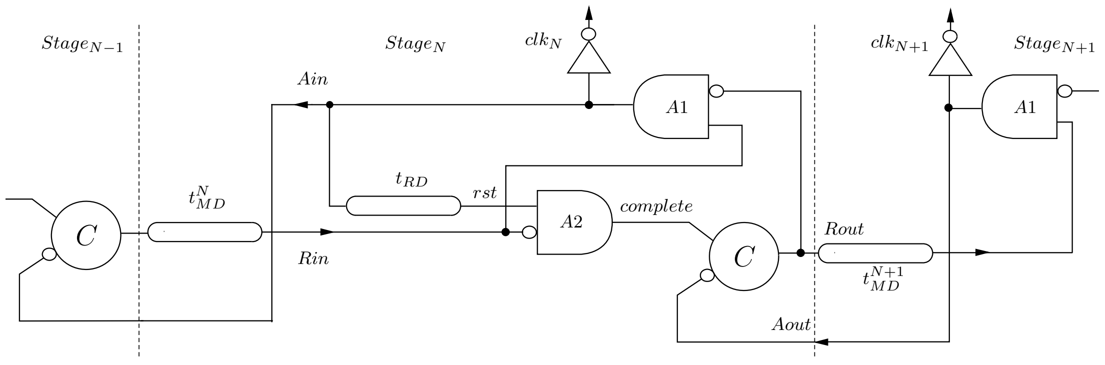


Fig. 8 Fastest environment for constraint analysis.

2.3 Timing Constraints

First, we turn to the timing constraints required for the desired operation described in Section 2.2. For constraint analysis, we assume that our controller is in a middle stage of a pipeline and that its environment (i.e., the controllers in the previous and next stages) operates at a speed equal to or slower than that of our controller. This is because we consider that the linear controller is the fastest, and assuming that the environment is 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.

The Signal Transition Graph (STG) for our controller for constraint analysis is depicted in Fig. 9. Thick arrows indicate the signal transitions generated from the environment of the controller whereas regular arrows indicate transitions made by the controller. Transitions are labelled with their associated gate delays. Note that, according to our assumptions, the environment delays are either equal to or larger than the delays incurred from the two similar controllers in the previous and next stages as shown in Fig. 8.

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

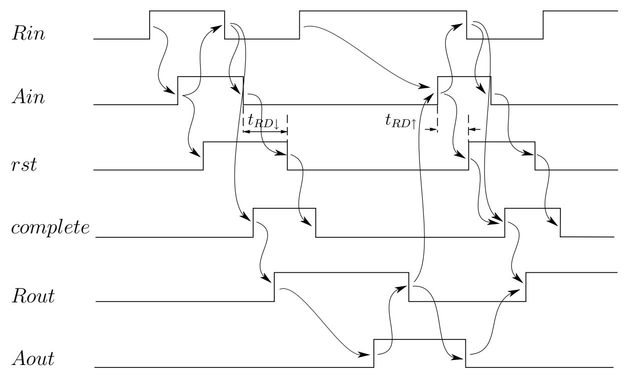


Fig. 7 Controller operation.

In the timing calculations, the inverted inputs of AND gates A1, A2 and of the C-element are not considered separately. They are attributed to the total delay of the gate.

Constraint 1. The first constraint imposes conditions to prevent data overwriting. In our controller, the pending output request ($Rout$ high) blocks any new requests on Rin. This requires $Rout$ to go high *before* the a new request (Rin high) is received. Thus the timing constraint can be formulated as follows:

$$t_{Rin\downarrow\to Rin\uparrow} \geq t_{Rin\downarrow\to Rout\uparrow}. \tag{C2}$$

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

$$t_{Rin\downarrow\to Rin\uparrow} = t_{AND\downarrow} + t_{Ain\downarrow\to Rin\uparrow}. \tag{P3}$$

The above path is labelled Ⓐ in Fig. 9. Note that $Ain\downarrow$ is always caused by $Rin\downarrow$ through AND gate *A1*. 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).

$$t_{Ain\downarrow\to Rin\uparrow} \geq t_{C\uparrow} + t^N_{MD\uparrow}. \tag{P4}$$

Thus, (P3) can be rewritten as

$$t_{Rin\downarrow\to Rin\uparrow} \geq t_{AND\downarrow} + t_{C\uparrow} + t^N_{MD\uparrow}. \tag{P5}$$

As for the right-hand side of (C2), we need to consider two cases where different events cause $Rout\uparrow$.

Case 1: If $Aout\downarrow$ is early enough compared with the next $complete\uparrow$ such that $Rout\uparrow$ is caused by $complete\uparrow$, then the following holds:

$$t_{Rin\downarrow\to Rout\uparrow} = max(0, -t_{Ain\uparrow\to Rin\downarrow} + t_{RD\uparrow}) + t_{AND\uparrow} + t_{C\uparrow}. \tag{P6}$$

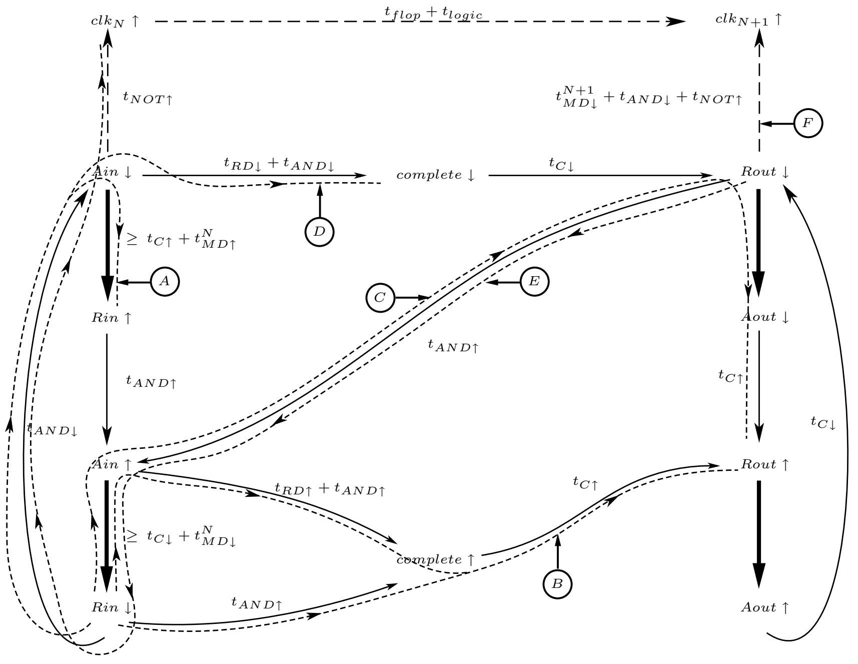


Fig. 9 STG for EA controller (constraint paths).

The above expression is labelled Ⓑ in Fig. 9. The max operator is used to get the larger of the delays from two concurrent paths. The first path corresponds to the delays from the input side, and the second path comprises delays local to the controller in the self-resetting loop. Since the second path actually originates 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,

$$t_{Ain\uparrow\rightarrow Rin\downarrow} \geq t_{C\downarrow} + t^N_{MD\downarrow} \tag{P7}$$

holds (see Fig. 9). Thus, (P6) can be rewritten as:

$$\begin{aligned} t_{Rin\downarrow\rightarrow Rout\uparrow} \leq & max(0, -(t_{C\downarrow} + t^N_{AND\downarrow}) + t_{RD\uparrow}) \\ & + t_{AND\uparrow} + t_{C\uparrow}. \end{aligned} \tag{P8}$$

From (P5) and (P8), a conservative version of the constraint (C2) is obtained in the form of constraints for the variable parameter $t^N_{MD\uparrow}$, the matched delay to be inserted between two stages of the pipeline, as follows:

$$\begin{aligned} & t_{AND\downarrow} + t_{C\uparrow} + t^N_{MD\uparrow} \geq t_{AND\uparrow} + t_{C\uparrow} \\ & \text{that is,} \qquad t^N_{MD\uparrow} \geq t_{AND\uparrow} - t_{AND\downarrow} \end{aligned} \tag{C9}$$

and

$$\begin{aligned} & t_{AND\downarrow} + t_{C\uparrow} + t^N_{MD\uparrow} \geq -(t_{C\downarrow} + t^N_{MD\downarrow}) + t_{RD\uparrow} \\ & \qquad\qquad + t_{AND\uparrow} + t_{C\uparrow} \\ & \text{that is,} \qquad t^N_{MD\uparrow} \geq t_{AND\uparrow} + t_{OR\uparrow} \\ & \qquad\qquad - (t_{C\downarrow} + 2 \cdot t_{AND\downarrow}). \end{aligned} \tag{C10}$$

In (C10), the occurrences of $t^N_{MD\downarrow}$ and $t_{RD\uparrow}$ have already been replaced with the equivalent gate delays $t_{AND\downarrow}$ and $t_{OR\uparrow}$ respectively.

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

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

The above expression is labelled Ⓒ in Fig. 9. From the delay assumption (P7), it can be rewritten as:

$$\begin{aligned} t_{Rin\downarrow\rightarrow Rout\uparrow} \leq & -(t_{C\downarrow} + t^N_{MD\downarrow} + t_{AND\uparrow}) \\ & + t_{Rout\downarrow\rightarrow Aout\downarrow} + t_{C\uparrow}. \end{aligned} \tag{P12}$$

From (P5) and (P12), another conservative version of the constraint (C2) for $t^N_{MD\uparrow}$ is obtained as follows:

$$\begin{aligned} & t_{AND\downarrow} + t_{C\uparrow} + t^N_{MD\uparrow} \geq -(t_{C\downarrow} + t^N_{MD\downarrow} + t_{AND\uparrow}) \\ & \qquad\qquad + t_{Rout\downarrow\rightarrow Aout\downarrow} + t_{C\uparrow} \\ & \text{that is,} \qquad t^N_{MD\uparrow} \geq t_{Rout\downarrow\rightarrow Aout\downarrow} - (t_{C\downarrow} \\ & \qquad\qquad + t_{AND\uparrow} + 2 \cdot t_{AND\downarrow}). \end{aligned} \tag{C13}$$

All the constraints derived for $t^N_{MD\uparrow}$ in Cases 1 and 2 (i.e., (C9), (C10), and (C13)) can be satisfied in the preferred application of our controller where there are processing elements within the pipeline and hence the matched delay $t^N_{MD\uparrow}$ is large enough to meet the above constraints.

Constraint 2. The next is a timing constraint to be satisfied by the self resetting delay. The *complete* signal should not be self-reset *before* the *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

$$t_{Rin\downarrow \to complete\downarrow} \geq t_{Rin\downarrow \to Rout\uparrow}. \tag{C14}$$

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

$$\begin{aligned} t_{Rin\downarrow \to complete\downarrow} &= t_{AND\downarrow} + t_{RD\downarrow} + t_{AND\downarrow} \\ &= t_{RD\downarrow} + 2 \cdot t_{AND\downarrow}. \end{aligned} \tag{P15}$$

The path of the above expression is labelled Ⓓ in Fig. 9. The right-hand side of the above constraint is the same as that of (C2). Thus, exactly the same two cases as those shown for Constraint 1 are considered, and the following three constraints are obtained for (C14).

Case 1: From (P15) and (P8), a conservative version of the constraint (C14) is obtained as follows:

$$\begin{aligned} & t_{RD\downarrow} + 2 \cdot t_{AND\downarrow} \geq t_{AND\uparrow} + t_{C\uparrow} \\ \text{that is,} \quad & t_{RD\downarrow} \geq t_{AND\uparrow} + t_{C\uparrow} - 2 \cdot t_{AND\downarrow} \end{aligned} \tag{C16}$$

and

$$\begin{aligned} & t_{RD\downarrow} + 2 \cdot t_{AND\downarrow} \geq -(t_{C\downarrow} + t^{N}_{MD\downarrow}) + t_{RD\uparrow} \\ & \qquad + t_{AND\uparrow} + t_{C\uparrow} \\ \text{that is,} \quad & t_{RD\downarrow} \geq t_{OR\uparrow} + t_{AND\uparrow} + t_{C\uparrow} \\ & \qquad - (t_{C\downarrow} + 3 \cdot t_{AND\downarrow}). \end{aligned} \tag{C17}$$

Case 2: From (P15) and (P12), another conservative version of constraint (C14) for $t_{RD\downarrow}$ is obtained as follows:

$$\begin{aligned} & t_{RD\downarrow} + 2 \cdot t_{AND\downarrow} \geq -(t_{C\downarrow} + t^{N}_{MD\downarrow} + t_{AND\uparrow}) \\ & \qquad + t_{Rout\downarrow \to Aout\downarrow} + t_{C\uparrow} \\ \text{that is,} \quad & t_{RD\downarrow} \geq t_{Rout\downarrow \to Aout\downarrow} + t_{C\uparrow} \\ & \qquad - (t_{AND\uparrow} + t_{C\downarrow} + 3 \cdot t_{AND\downarrow}). \end{aligned} \tag{C18}$$

The constraints derived for $t_{RD\downarrow}$ in Cases 1 and 2 (i.e., (C16), (C17), and (C18)) should be considered when selecting the minimum delay for the self-resetting loop.

Constraint 3. The last timing constraint prevents $Stage_N$ clock (clk_N) from capturing new data *before* $Stage_{N+1}$ captures data already processed between two stages. When we use $Rout\downarrow$ event, which signals the arrival of new data to the $Stage_{N+1}$, and unblocks the requests pending at AND gate $A1$, as the starting point of time measurements, this timing constraint can be formulated as follows.

$$t_{Rout\downarrow \to clk_N\uparrow} \geq t_{Rout\downarrow \to clk_{N+1}\uparrow}. \tag{C19}$$

Right-hand side of the above constraint (labelled Ⓕ in Fig. 9), is the path from the $Rout$ falling edge to the capture of data by the clock clk_{N+1}. The left-hand side (labelled Ⓔ in Fig. 9), is the path from $Rout$ falling edge to the capture of new data from clk_N.

$$\begin{aligned} t_{Rout\downarrow \to clk_N\uparrow} = \; & t_{AND\uparrow} + t_{Ain\uparrow \to Rin\downarrow} \\ & + t_{AND\downarrow} + t_{NOT\uparrow} \\ \geq \; & t_{AND\uparrow} + t_{C\downarrow} + t^{N}_{MD\downarrow} \\ & + t_{AND\downarrow} + t_{NOT\uparrow}. \end{aligned} \tag{P20}$$

For the worst case of the constraint (C19), the equality of the property (P20) should hold. In other words, the input environment is the fastest possible (i.e., delays are equal to the those incurred by the linear controller). This gives an upper bound for the right-hand side of the constraint.

$$\begin{aligned} t_{Rout\downarrow \to clk_{N+1}\uparrow} \leq \; & t_{AND\uparrow} + t_{C\downarrow} + t^{N}_{MD\downarrow} \\ & + t_{AND\downarrow} + t_{NOT\uparrow}. \end{aligned} \tag{C21}$$

In the case of linear controller in the $Stage_{N+1}$, from Fig. 9, the following holds.

$$t_{Rout\downarrow \to clk_{N+1}\uparrow} = t^{N+1}_{MD\downarrow} + t_{AND\downarrow} + t_{NOT\uparrow}. \tag{P22}$$

Hence, in the linear pipeline, the above environment delay satisfies (C21) because, when the equations (C21) and (P22) are simplified replacing $t^{N}_{MD\downarrow}$ and $t^{N+1}_{MD\downarrow}$ with constant $t_{AND\downarrow}$ delay, the right-hand sides of the resulting expressions amounts to 5 gate delays and 3 gate delays, respectively.

The controller is model checked using UPPAAL model checker [20, 21] tool to verify the correctness of the functionality and to conclude that the above constraints comprehensively guarantee the operation of the controller.

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 show which components of the latter performance metric can be hidden in the case of a pipeline with logic processing where the EA protocol has a competitive edge. We assume that the controller in a middle stage of a pipeline with the similar 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, we can derive the maximum performance of our controller.

The Fig. 10 depicts the STG for our controller in desired operation, when it meets the above constraints. Here, the environment delays are equal to the delays incurred from two similar controllers in the previous and next stages according to our second assumption. Dashed-line 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 these stages, which are not directly in the control path of the main control logic but are 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.

The cycle time is defined as the interval between two successive data items passing through a pipeline stage when

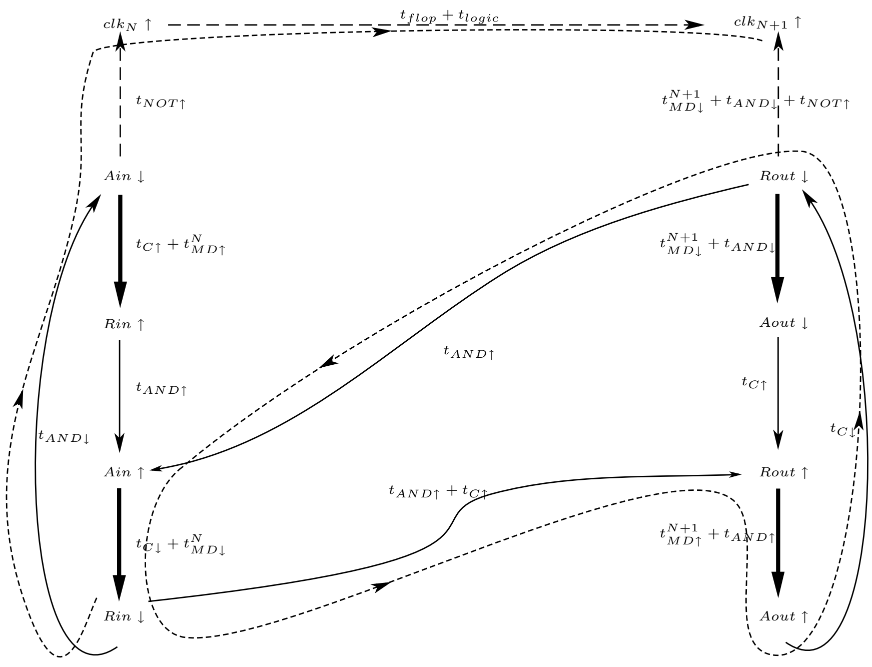


Fig. 10 STG for EA controller.

the pipeline is operating at maximum speed. For this purpose, we can measure the gate delays between two successive *clk* rising edges or equivalently the delay between two successive falling edges of *Rin*.

First, we identify the controller's critical cycle using the STG branch and merge points. This critical cycle lies on the path ($Rin \downarrow \rightarrow Rout \uparrow \rightarrow Aout \uparrow \rightarrow Rout \downarrow \rightarrow Ain \uparrow \rightarrow Rin \downarrow$) indicated by the cycle composed of short dashes. In fact, it is necessary to unfold the STG to the 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 controller's cycle. Details of the inductive proof that arrives at the same conclusion have been omitted. The cycle time can be obtained from the critical path as a function of gate delays and required matched delays (t^N_{MD} and t^{N+1}_{MD}) as follows.

$$T = 3 \cdot t_{AND\uparrow} + 2 \cdot t_{C\downarrow} + t_{C\uparrow} + t^{N+1}_{MD\uparrow} + t^{N}_{MD\downarrow}. \quad (23)$$

Here, all terms except $t^{N+1}_{MD\uparrow}$ are constant gate delays. To obtain the cycle time and forward latency in terms of logic processing delay (t_{logic}), we need to express the required matched delay t^{N+1}_{MD} for the operations in terms of t_{logic}. When the data is captured with $clk_N \uparrow$, the next stage clock $clk_{N+1} \uparrow$ needs to be made after a delay of $t_{flop} + t_{logic}$, where t_{flop} is the delay of the date register. We can relate t_{logic} to t^{N+1}_{MD} by measuring the same delay in two paths to the event of $clk_{N+1} \uparrow$.

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

$$T_1 = t_{AND\uparrow} + t_{C\uparrow} + t^{N+1}_{MD\uparrow} + t_{AND\uparrow} + t_{C\downarrow} + t^{N}_{MD\downarrow} + t_{AND\downarrow} + t_{NOT\uparrow}. \quad (24)$$

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

$$T_2 = t_{AND\downarrow} + t_{NOT\uparrow} + t_{flop} + t_{logic}. \quad (25)$$

To ensure the correct operation of the pipeline, $T_1 \geq T_2$ must hold. Thus, from the above two equations, we can derive an expression for the minimum value of $t^{N+1}_{MD\uparrow}$ as

$$t^{N+1}_{MD\uparrow} \geq (t_{flop} + t_{logic}) - (2 \cdot t_{AND\uparrow} + t^{N}_{MD\downarrow} + t_{C\uparrow} + t_{C\downarrow}). \quad (26)$$

Thus, if

$$t_{logic} \geq (2 \cdot t_{AND\uparrow} + t^{N}_{MD\downarrow} + t_{C\uparrow} + t_{C\downarrow}) - t_{flop} \quad (27)$$

holds, we can find the cycle time in terms of t_{logic} by replacing $t^{N+1}_{MD\uparrow}$ in equation (23) by the right-hand side of (26). The the cycle time for the linear controller of EA protocol can be expressed as follows.

$$T^{l}_{EA} = t_{flop} + t_{logic} + t_{AND\uparrow} + t_{C\downarrow}. \quad (28)$$

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

If the logic processing time is smaller and the inequality (27) does not hold, we obtain the minimum cycle time (maximum throughput) of this controller directly from equation (23) by using $t_{MD\uparrow}^{N+1} = t_{MD\downarrow}^{N} = 0$, that is

$$T_{EA}^{l}|_{min} = 3 \cdot t_{AND\uparrow} + 2 \cdot t_{C\downarrow} + t_{C\uparrow}. \qquad (29)$$

The above cycle minimum time is valid because it is possible to remove the matched delay without violating the timing constraints derived for $t_{MD\uparrow}^{N}$[†]. This could be confirmed in our experiments 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 $Rin\downarrow$ of the STG is shown in Fig. 10 by the line of short dashes. When the inequality (27) holds, we can use a similar argument to obtain the forward latency as follows.

$$L_{EA}^{l} = t_{AND\downarrow} + t_{NOT\uparrow} + t_{flop} + t_{logic}. \qquad (30)$$

When the logic processing delay is small and inequality (27) does not hold, the critical path for forward latency lies on the path: $Rin \downarrow \rightarrow Rout \uparrow \rightarrow Aout \uparrow \rightarrow Rout \downarrow \rightarrow clk_{N+1} \uparrow$, which is:

$$\begin{aligned} L = {} & t_{AND\uparrow} + t_{C\uparrow} + t_{MD\uparrow}^{N+1} + t_{AND\uparrow} + t_{C\downarrow} \\ & + t_{MD\downarrow}^{N+1} + t_{AND\downarrow} + t_{NOT\uparrow}. \qquad (31) \end{aligned}$$

Like the minimum cycle time, the minimum forward latency on this path can be derived using $t_{MD\uparrow}^{N+1} = t_{MD\downarrow}^{N+1} = 0$. It is given by

$$L_{EA}^{l}|_{min} = 2 \cdot t_{AND\uparrow} + t_{AND\downarrow} + t_{C\uparrow} + t_{C\downarrow} + t_{NOT\uparrow}. \quad (32)$$

In a general pipeline with logic processing, condition (27) often holds. In that case, cycle time and forward latency for our controller are given by equations (28) and (30), respectively.

3. Comparison with 2- and 4-phase Pipeline Controllers

To demonstrate the advantage of the EA-protocol-based controller, we compared its performance with 2- and 4-phase pipeline controllers. The following subsections describe the controllers used for this comparison and their key features.

3.1 2-phase Controller: MOUSETRAP

For the 2-phase or the transition signalling protocol, the MOUSETRAP controller was selected for its simplicity and high performance. As shown in Fig. 11, this controller consists of a simple *transparent latch* (denoted by the rectangular box) and an XNOR gate. The signal $enable$ is used to drive data latches (instead of D-flipflops) in the data path.

Initially, all control signals are low except for the $enable$ signals that make all the pipeline stages transparent. When the first data item flows through the pipeline stage, it flips the values of Rin_N, $Rout_N$, and Ain_N exactly once (to high). Subsequently, the second data item flips all these signals once again (to low). This is 2-phase (or transition) signalling where each transition (either up or down) indicates a distinct request or acknowledgement.

Once a data item has been captured by stage latches, three actions occur in parallel: (i) the request to the next-stage $Rout_N$ is made; (ii) an acknowledgement, Ain_N, is sent to the previous stage, allowing the next data item to be sent; and finally (iii) $enable_N$ is lowered to make the stage latches opaque, protecting the current data from being overwritten. Subsequently, when an acknowledgement, $Aout_N$ is received from $stage_{N+1}$, the latch in $stage_N$ is re-enabled (i.e., made transparent).

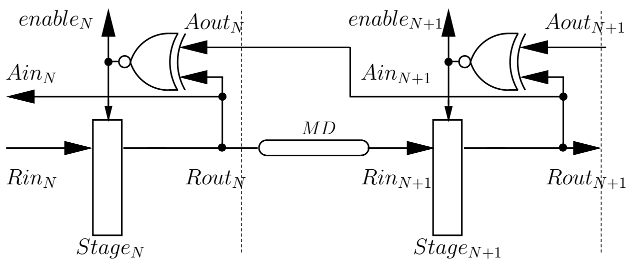


Fig. 11 MOUSETRAP controller.

In [4] the operation of the MOUSETRAP controller in both high-speed pipelines and pipelines with logic processing is described in detail. The authors also presented the controller's cycle time and the forward latency. The most important point to note regarding the EA protocol and 4-phase signalling protocol is that there is no resetting overhead in the 2-phase protocol, hence there is none controller either. We have extended the original derivation of cycle time and latency for MOUSETRAP to cases with and without logic processing. The complete derivation can be found in Appendix A. The results are summarized as follows.

$$T_{2P}^{l} = 2 \cdot t_{latch} + t_{logic} + t_{XNOR\uparrow} \qquad (33)$$

$$L_{2P}^{l} = t_{latch} + t_{logic}. \qquad (34)$$

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

$$T_{2P}^{l}|_{min} = 2 \cdot t_{latch} + t_{XNOR\uparrow} \qquad (35)$$

$$L_{2P}^{l}|_{min} = t_{latch}. \qquad (36)$$

3.2 4-phase Controller

We used the 4-phase controller proposed in [17] for this comparison. That controller is shown in Fig. 12. *G1* and *G2* are complex gates composing the controller.

We could derive the cycle time and latency for this 4-phase controller using a similar mechanism to the one used

[†] In the pipeline, the constraints derived for t_{MD}^{N} of $stage_N$ are valid for t_{MD}^{N+1} of $stage_{N+1}$

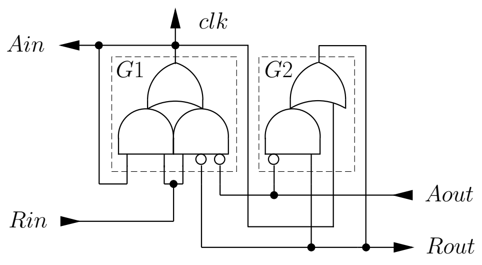


Fig. 12 4-phase controller.

in the EA protocol controller. The formal analysis for obtaining them is described in Appendix B. The results can be summarized as follows. When t_{logic} is large enough that the inequality

$$t_{logic} \geq t_{G1\uparrow} + t_{G2\uparrow} - t_{flop} \tag{37}$$

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

$$\begin{aligned} T^l_{4P} &= t_{flop} + t_{logic} + t_{G1\uparrow} + t_{G1\downarrow} \\ &\quad + t_{G2\downarrow} + t_{AND\downarrow} \end{aligned} \tag{38}$$

$$L^l_{4P} = t_{flop} + t_{logic} + t_{G1\uparrow}. \tag{39}$$

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

$$T^l_{4P}|_{min} = 2 \cdot t_{G1\uparrow} + t_{G2\uparrow} + t_{G1\downarrow} + t_{G2\downarrow} \tag{40}$$

$$L^l_{4P}|_{min} = 2 \cdot t_{G1\uparrow} + t_{G2\uparrow}. \tag{41}$$

3.3 Performance Comparison

The merits of using the EA controller could be observed in the case where t_{logic} satisfies the condition derived in (27). Then, the cycle time for our controller is given by (28). This can be compared analytically with the 2- and 4-phase protocols using equations (33) and (38). It is not possible to compare the cycle times without specific delays obtained from technology libraries. However, we can get an idea of the controller overhead in the overall cycle time in each case. Note that the data-path delay for our controller and for the 4-phase controller is both $t_{flop} + t_{logic}$ because, we use D-flipflops on the data-path. In the case of the MOUSETRAP controller, the data-path delay is $t_{latch} + t_{logic}$ as a result of the use of transparent latches. Any additional terms appearing in the cycle-time expressions apart from these 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 the 4-phase controller, the overhead is 4 gate delays, which is incurred by the resetting period.

However, our controller does not exhibit the performance advantage in first-in-first-out (FIFO) types of pipelines, which have no logic processing. In that case, its minimum cycle time is given by (29), where the controller overhead is exposed in the controller's critical cycle. Compared with the 2-phase controller (35), this is clearly larger.

The forward latencies of the three controllers can be compared using equations (30), (34), and (39). The MOUSETRAP controller clearly exhibits the lowest forward latency, whereas the 4-phase controller shows slightly lower latency than our controller, given that the gate delays are equal. The forward latencies when there is no logic processing are given by equations (32), (36), and (41). Again, we can see that the MOUSETRAP controller has the lowest latency and our controller has the highest.

It should be noted that neither our controller nor the MOUSETRAP controller is a Speed Independent(SI) circuit whereas the 4-phase controller is. SI circuits work correctly under the unbounded gate delay model, for which gate delays are unbounded (yet finite) and wire delays are zero. In our controller (and also in the MOUSETRAP), there are timing constraints that should be held by the gate delays of the selected technology for the design. The performance advantages of both controllers over the 4-phase controller depends partly depend this as well.

4. Conditional Branch (CB) Controller

We have used the CB non-linear pipeline operation to demonstrate the simplicity of the EA protocol (which is a return-to-zero protocol) in composing complex pipeline constructs. The CB operation implements the logical equivalent of an *if-then-else* construct. The data in the input is diverted to *one* of the branches depending on the *select* signal to the controller.

The CB controller communicates with the input stage by means of *Rin* and *Ain* signals, whereas the two output stage control signals are *Rout1*, *Aout1* and *Rout2*, *Aout2*, respectively as shown in Fig. 13. When a request *Rin* is made from the previous stage of the pipeline, data is captured by the *clk* signal. Acknowledgement *Ain* is sent to the input stage when the data is captured. Depending on the *select* signal, the request is routed on either the first branch *Rout1* or the second branch *Rout2*. It is assumed that the *select* signal is generated from one or more data-path signals.

4.1 Early Acknowledgement CB Controller

The CB controller for the EA protocol is a simple extension of its linear controller. The controller can be composed of a linear controller (for EA protocol), demultiplexer, delay element (SD), and an OR gate. The CB controller at $Stage_N$ of a pipeline is shown in Fig. 13. The *Rin* and *Ain* of the controller are handled by the linear controller used within the CB controller. A function generator *gen* that produces *select* from $stage_N$ data is explicitly considered for analyzing constraints imposed by such an application. In Fig. 13, the rectangular box connected to clk_N signal represents the $stage_N$ register which consists of D-flipflops. $data'_N$ is the captured data of $data_N$ in the stage register

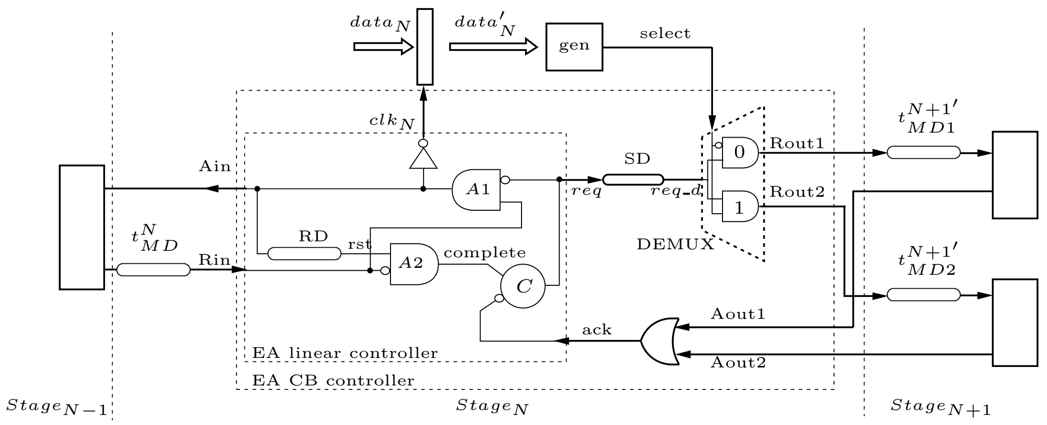


Fig. 13 Early Acknowledgement CB controller.

according to our naming convention. The *select* signal is generated using function generator *gen* from this captured data as shown. The asymmetric delay element 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 SD for this correct sampling of the *select* signal is presented in the constraint analysis of this controller. The *select* signal diverts the delayed request *req_d* from the linear controller to either the *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 the *ack* to the linear controller.

4.1.1 Timing Constraints

Timing constraints for the CB controller are analyzed as an extension of the linear controller constraints presented in Section 2.3. Again, we obtain timing constraints to satisfy the desired operation described in Section 4 assuming that the CB controller is in a middle stage of a pipeline with the environment operating at a speed equal to or slower than our linear controller. First, the three constraints presented in Section 2.3 are reevaluated to ensure proper operation of the linear controller used within the CB controller. Then, an additional constraint on SD delay element for proper operation of demultiplexer is presented.

Constraint 1 and 2. The CB controller is viewed as a linear controller (or linear pipeline) from the input side because, it uses a linear EA controller to communicate with *Rin* and *Ain*. The difference can be perceived only when viewed from the controller's branches owing to the additional branching logic for request and acknowledge. Thus, constraints involving $t_{Rout\downarrow \to Aout\downarrow}$ should be reconsidered. This corresponds to **Case 2** of each constraint, where $Aout\downarrow$ causes $Rout\uparrow$. The two constraints (C13) and (C18) can be restated for the linear controller within the CB controller using the labels *req* and *ack* for the output side as in Fig. 13.

$$t^N_{MD\uparrow} \geq t_{req\downarrow\to ack\downarrow} - (t_{C\downarrow} + t_{AND\uparrow} + 2 \cdot t_{AND\downarrow}) \quad \text{(C42)}$$

and $$t_{RD\downarrow} \geq t_{req\downarrow\to ack\downarrow} + t_{C\uparrow} - (t_{AND\uparrow} + t_{C\downarrow} + 3 \cdot t_{AND\downarrow}) \quad \text{(C43)}$$

where

$$\begin{aligned} t_{req\downarrow\to ack\downarrow} &= t_{SD\downarrow} + t_{AND\downarrow} + t_{Rout1\downarrow\to Aout1\downarrow} + t_{OR\downarrow} \\ &= 2 \cdot t_{AND\downarrow} + t_{Rout1\downarrow\to Aout1\downarrow} + t_{OR\downarrow}. \end{aligned} \quad \text{(C44)}$$

Here, $t^N_{MD\uparrow}$ and $t_{RD\downarrow}$ should be selected to satisfy these new constraints.

Constraint 3. This constraint can be restated for CB controller from (C21) as follows.

$$t_{req\downarrow\to clk_{N+1}\uparrow} \leq t_{AND\uparrow} + t_{C\downarrow} + t^N_{MD\downarrow} + t_{AND\downarrow} + t_{NOT\uparrow}. \quad \text{(C45)}$$

This condition must be satisfied by any controller in the output stage.

For example, this constraint can be considered for two cases 1) when there is a linear controller at the output stage, and 2) when there is a CB controller at the output stage. It should be noted that there is no difference in these two cases as far as this constraint is concerned because, the input side of a CB controller is composed of a linear controller. Hence, the output stage, $Rout1/Aout1$ (and/or $Rout2/Aout2$) will be connected to a linear controller in either case. The path delay from $req\downarrow$ to $clk_{N+1}\uparrow$ (similar to (P22) in linear controller) in above two cases can be obtained from the STG diagram for the CB controller shown in Fig. 14.

$$t_{req\downarrow\to clk_{N+1}\uparrow} = [t_{SD\downarrow} + t_{AND\downarrow}] + t'_{MD1\downarrow} + t_{AND\downarrow} + t_{NOT\uparrow}. \quad \text{(P46)}$$

Due to the extra overhead incurred by the SD and demultiplexer, the property becomes a hard condition to satisfy with 5 gate delays in either side of the inequality. Satisfaction of

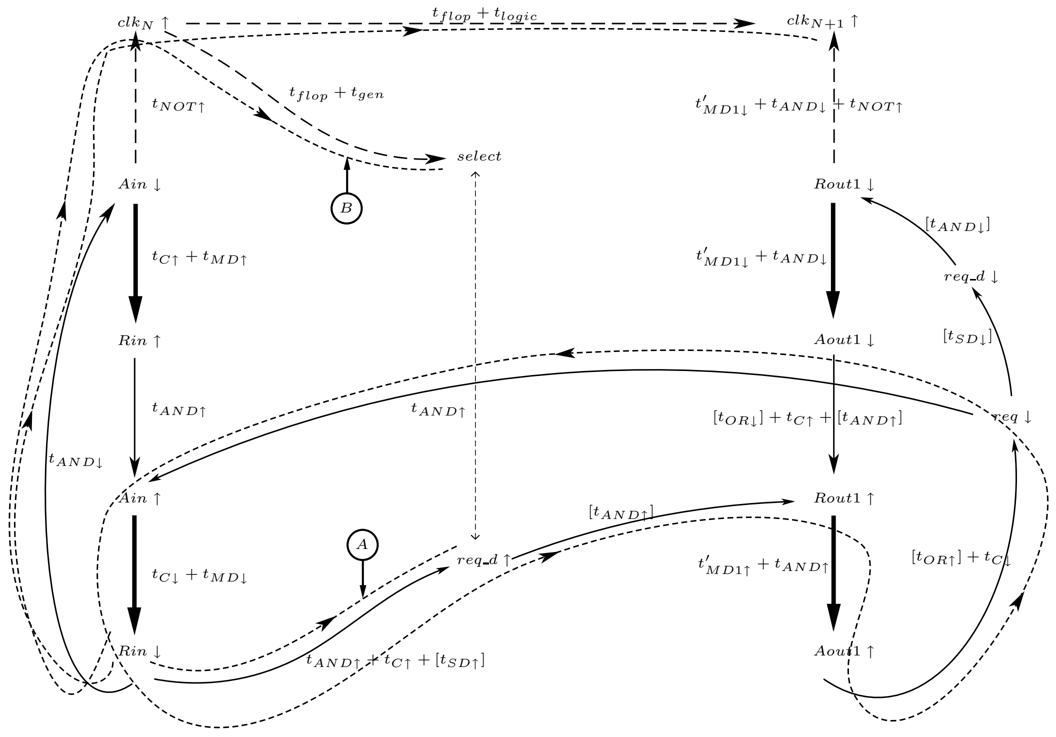


Fig. 14 STG for EA-CB controller.

this constraint can be guaranteed by increasing $t^N_{MD\downarrow}$ to be more than one gate delay at the expense of introducing additional overhead to the critical cycle of the controller. (This can be done by adding buffers in between the *out* port and the rightmost AND gate of the asymmetric delay in Fig. 3. As a result, it also increases the $t_{MD\uparrow}$ by the same amount of delay incurred by the buffers. It should be adjusted back by reducing the AND gates which comprises the variable part of the delay.)

Constraint 4. An additional constraint on a CB controller requires the *select* signal to be valid *before req_d* goes high. This ensures proper operation of the demultiplexer that switches the request *req* to either branch depending on the *select* signal. The EA protocol stipulates that the *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 becomes:

$$t_{Rin\downarrow\to req\_d\uparrow} \geq t_{Rin\downarrow\to select}. \tag{C47}$$

For $Rin\downarrow$ to cause $req\_d\uparrow$, at least $t_{AND\uparrow}$ of AND gate *A2*, $t_{C\uparrow}$ of the *C*-element, and $t_{SD\uparrow}$ of *SD* should occur (path labelled Ⓐ in Fig. 14). Hence, the lower bound for the left-hand side of the above constraint can be expressed as follows.

$$t_{Rin\downarrow\to req\_d\uparrow} \geq t_{AND\uparrow}+t_{C\uparrow}+t_{SD\uparrow}. \tag{P48}$$

The $Rin\downarrow$ also causes data capture by lowering Ain though the AND gate *A1* and raising the clock NOT gate. The function generator produces the correct *select* signal from the captured data after time t_{gen} has elapsed (path labelled Ⓑ in Fig. 14). Hence, the right-hand side of the constraint can be expressed as

$$t_{Rin\downarrow\to select} = t_{AND\downarrow}+t_{NOT\uparrow}+t_{flop}+t_{gen}. \tag{P49}$$

Thus, the constraint on the asymmetric delay can be derived as

$$\begin{aligned} t_{AND\uparrow}+t_{C\uparrow}+t_{SD\uparrow} &\geq t_{AND\downarrow}+t_{NOT\uparrow}+t_{flop}+t_{gen} \\ t_{SD\uparrow} &\geq t_{flop}+t_{gen}+t_{AND\downarrow}+t_{NOT\uparrow} \\ &\quad -(t_{AND\uparrow}+t_{C\uparrow}). \end{aligned} \tag{C50}$$

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

4.1.2 Performance

In this section, the controller's cycle time and forward latency are derived analytically as was done for the linear controller. The 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 (27).

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

The notable difference from the linear controller's STG is in the matched delays $MD1$ and $MD2$ for the controller's output branches. As shown in Fig. 13 and detailed in the constraint analysis, part of the output-side matched delays may be used for SD inside our controller to compensate for the select generator function. The matched delays outside the controller $MD1'$ and $MD2'$ are selected such that the original matched delay remains the same. i.e.,

$$\begin{aligned} & t_{MD1\uparrow} = t'_{MD1\uparrow} + t_{SD\uparrow} \\ \text{and} \quad & t_{MD2\uparrow} = t'_{MD2\uparrow} + t_{SD\uparrow}. \end{aligned}$$

We can measure the delays in the control cycle and data cycle to derive the cycle time in t_{logic} and the inequality for optimal controller operation. The cycle time (indicated by the cycle composed of short dashes in Fig. 14) in terms of gate delays can be expressed as follows.

$$\begin{aligned} T = {} & t_{AND\uparrow} + t_{C\uparrow} + [t_{SD\uparrow}] + [t_{AND\uparrow}] \\ & + t'_{MD1\uparrow} + t_{AND\uparrow} + [t_{OR\uparrow}] + t_{C\downarrow} \\ & + t_{AND\uparrow} + t_{C\downarrow} + t_{MD\downarrow}. \end{aligned} \tag{51}$$

The delays enclosed within square brackets indicate the extra delays of the path due to CB operation. We can express this cycle time in terms of t_{logic} by measuring the delays in the control and data paths.

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

$$\begin{aligned} T_{CB1} = {} & t_{AND\uparrow} + t_{C\uparrow} + [t_{SD\uparrow}] + [t_{AND\uparrow}] \\ & + t'_{MD1\uparrow} + t_{AND\uparrow} + [t_{OR\uparrow}] + t_{C\downarrow} + [t_{SD\downarrow}] \\ & + [t_{AND\downarrow}] + t'_{MD1\downarrow} + t_{AND\downarrow} + t_{NOT\uparrow}. \end{aligned} \tag{52}$$

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

$$T_{CB2} = t_{AND\downarrow} + t_{NOT\uparrow} + t_{flop} + t_{logic}. \tag{53}$$

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

$$\begin{aligned} t'_{MD1\uparrow} \geq {} & (t_{flop} + t_{logic}) \\ & - (2 \cdot t_{AND\uparrow} + t_{C\uparrow} + t_{C\downarrow} + t'_{MD1\downarrow}) \\ & - [t_{SD\uparrow} + t_{AND\uparrow} + t_{AND\downarrow} + t_{OR\uparrow} + t_{SD\downarrow}]. \end{aligned} \tag{54}$$

Thus, if

$$\begin{aligned} t_{logic} \geq {} & (2 \cdot t_{AND\uparrow} + t_{C\uparrow} + t_{C\downarrow} + t'_{MD1\downarrow} \\ & + [t_{SD\uparrow} + t_{AND\uparrow} + t_{AND\downarrow} + t_{OR\uparrow} \\ & + t_{SD\downarrow}]) - t_{flop} \end{aligned} \tag{55}$$

holds, the cycle time for CB controller T^{cb}_{EA} can be expressed in terms of t_{logic} by substituting the minimum of (54) into equation (51).

$$T^{cb}_{EA} = t_{flop} + t_{logic} + t_{AND\uparrow} + t_{C\downarrow} - [2.t_{AND\downarrow}]. \tag{56}$$

To simplify the above expression, we use the fact that $t'_{MD1\downarrow} = t_{MD\downarrow} = t_{SD\downarrow} = t_{AND\downarrow}$ in accordance with our implementation.

Note that in inequality (55), the additional delays due to CB operation (enclosed in brackets) constitute two parts. 1) t_{SD}, which compensates for t_{gen} and 2) delays due to the DEMUX and the OR gate. When the condition in (55) is satisfied, the controller's cycle time is given by (56). According to inequalities of (27) and (55), the minimum t_{logic} required to hide the additional overhead incurred by the CB controller is higher than that for the linear controller.

In the case where the logic processing time is too small and inequality (55) does not hold, we get the minimum cycle time directly from the equation (51) with $t'_{MD1\uparrow} = t_{MD\downarrow} = 0$. It is:

$$\begin{aligned} T^{cb}_{EA}|_{min} = {} & 3 \cdot t_{AND\uparrow} + t_{C\uparrow} + 2 \cdot t_{C\downarrow} \\ & + [t_{SD\uparrow} + t_{AND\uparrow} + t_{OR\uparrow}]. \end{aligned} \tag{57}$$

Forward latency is also measured in a manner similar to that for the linear controller; this is marked in dashed lines on the STG diagram. For sufficiently large t_{logic}, forward latency has the same terms (despite the minimum t_{logic} being larger) as for the linear controller. That is,

$$L^{cb}_{EA} = t_{AND\downarrow} + t_{NOT\uparrow} + t_{flop} + t_{logic}. \tag{58}$$

When t_{logic} is small and the inequality (55) does not hold, the critical path lies on the path: $Rin \downarrow\rightarrow req\_d \uparrow\rightarrow Rout_1 \uparrow\rightarrow Aout_1 \uparrow\rightarrow req \downarrow\rightarrow Rout_1 \downarrow\rightarrow clk_{N+1} \uparrow$.

$$\begin{aligned} L = {} & t_{AND\uparrow} + t_{C\uparrow} + [t_{SD\uparrow}] + [t_{AND\uparrow}] + t'_{MD1\uparrow} \\ & + t_{AND\uparrow} + [t_{OR\uparrow}] + t_{C\downarrow} + [2.t_{AND\downarrow}] \\ & + t'_{MD1\downarrow} + +t_{AND\downarrow} + t_{NOT\uparrow}. \end{aligned} \tag{59}$$

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.

$$\begin{aligned} L^{cb}_{EA}|_{min} = {} & 2 \cdot t_{AND\uparrow} + t_{AND\downarrow} + t_{C\uparrow} + t_{C\downarrow} + t_{NOT\uparrow} \\ & + [t_{SD\uparrow} + t_{AND\uparrow} + 2.t_{AND\downarrow} + t_{OR\uparrow}]. \end{aligned} \tag{60}$$

4.2 2-phase CB Controller

The CB controller for the transition signalling protocol is not as straightforward as in the EA 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. The CB controller for 2-phase protocol based on [22] is shown in Fig. 15. Note that D-flops are used instead of the transparent latches used in the MOUSETRAP controller for a linear pipeline because, the D-flipflop-based controller is more robust than the transparent-latch-based controller in this case.

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

Either request event causes the *complete* signal to go low, indicating that the latched data is being passed to the output stage, which effectively blocks new requests from the input side. Upon the acknowledgement of the corresponding branch, each pair of request and acknowledgement signals returns 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 the linear controller (MOUSETRAP), *s1* and *s2* the request-generating toggle flops and completion detection mechanism incur a considerable overhead in their operation, which adversely affects the controller's performance.

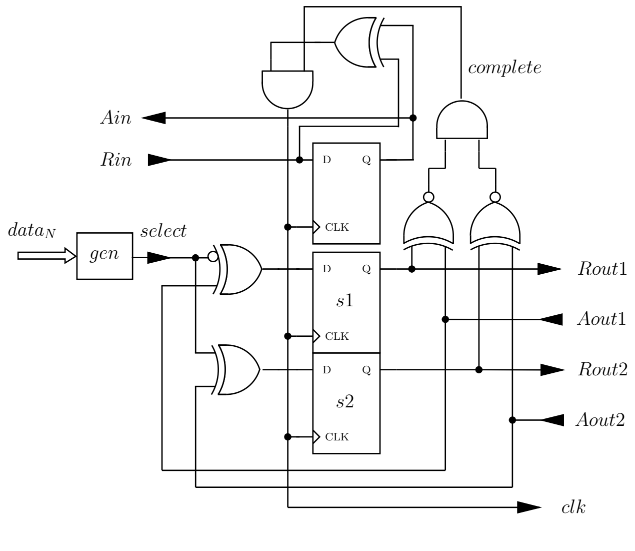


Fig. 15 2-phase CB controller.

A formal analysis to obtain this controller's cycle time is presented in Appendix C. We found that

$$\text{if} \quad t_{logic} \geq t_{latch} + t_{XNOR\downarrow} \tag{61}$$

holds, this controller's cycle time and forward latency can be obtained as:

$$T_{2P}^{cb} = t_{flop} + t_{logic} + 2 \cdot t_{AND\uparrow} + (t_{XNOR\uparrow} - t_{XNOR\downarrow}) \tag{62}$$

$$L_{2P}^{cb} = t_{flop} + t_{logic} + t_{XOR\uparrow} + t_{AND\uparrow}. \tag{63}$$

If the above condition does not hold, the minima of these two parameters are obtained as follows.

$$T_{2P}^{cb}|_{min} = t_{flop} + t_{latch} + t_{XNOR\uparrow} + 2 \cdot t_{AND\uparrow} \tag{64}$$

$$L_{2P}^{cb}|_{min} = t_{XOR\uparrow} + t_{AND\uparrow} + t_{flop} + t_{latch} + t_{XNOR\downarrow}. \tag{65}$$

4.3 4-phase CB Controller

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

The cycle time and forward latency of this controller are obtained in a way similar to that of CB controller for the EA protocol. Details are given in Appendix D. The obtained expressions can be summarized as follows.

$$\text{If} \quad t_{logic} \geq (t_{G1\uparrow} + t_{G2\uparrow} + [t_{SD\uparrow} + t_{AND\uparrow}]) - t_{flop}, \tag{66}$$

then

$$T_{4P}^{cb} = t_{flop} + t_{logic} + t_{G1\uparrow} + t_{G1\downarrow} + t_{G2\downarrow} + t_{AND\downarrow} + [2 \cdot t_{AND\downarrow} + t_{OR\uparrow} + t_{OR\downarrow}] \tag{67}$$

$$L_{4P}^{cb} = t_{flop} + t_{logic} + t_{G1\uparrow}. \tag{68}$$

Otherwise, the minimum cycle time and forward latency are

$$T_{4P}^{cb}|_{min} = 2 \cdot t_{G1\uparrow} + t_{G2\uparrow} + t_{G1\downarrow} + t_{G2\downarrow} + [t_{SD\uparrow} + t_{AND\uparrow} + 2 \cdot t_{AND\downarrow} + t_{OR\uparrow} + t_{OR\downarrow}] \tag{69}$$

$$L_{4P}^{cb}|_{min} = 2 \cdot t_{G1\uparrow} + t_{G2\uparrow} + [t_{SD\uparrow} + t_{AND\uparrow}]. \tag{70}$$

4.4 Performance Comparison

Again, an accurate comparison of cycle times requires specific delay values from technology libraries. However, we use employ the same mechanism to compare the controller overheads as we used in the the linear controller comparison (in Section 3.3). In cycle time comparison, we can observe from equations (56), (62) and (67) that 4-phase cycle time has a higher overhead than the 2-phase controller and our controller. Hence, we put more emphasis on comparing the first two controllers because this comparison shows the advantage of our controller over the 2-phase controller. For the EA controller (56) and 2-phase controller (62), an approximate comparison can be done assuming that $t_{AND\uparrow} \approx t_{AND\downarrow}$ and $t_{XNOR\uparrow} \approx t_{XNOR\downarrow}$, which further simplifies the cycle times to:

$$T_{EA}^{cb} = t_{flop} + t_{logic} + t_{C\downarrow} - t_{AND\downarrow} \tag{71}$$

$$T_{2P}^{cb} = t_{flop} + t_{logic} + 2 \cdot t_{AND\uparrow}. \tag{72}$$

A comparison of the controllers' simplified cycle times (72) and (71) shows that the latter is slightly better provided that

$$t_{C\downarrow} - t_{AND\downarrow} < 2 \cdot t_{AND\uparrow} \tag{73}$$

This can hold in the case of many technologies mainly owing to the fact that the right-hand side is two gate delays while the left-hand side is less than one gate delay. This gives a slight performance advantage to the EA-protocol-based controller. In a process technology where gate delays can be chosen such that $t_{AND\uparrow} < t_{AND\downarrow}$ and $t_{XNOR\uparrow} < t_{XNOR\downarrow}$ (for example, using transistor sizing in ASIC technologies), the cycle times in *both* cases can be reduced according to the expressions that we obtained ((56) and (62)). Again, this give rises to the above condition, which determines the higher performance of the two controllers. As shown in Section 5.2, in the case of our experiments on an FPGA where the gate delays are identical, we observed that the EA controller's cycle time was slightly better.

When there is no logic processing, the cycle times can be compared using equations (57), (64), and (69). With the minimum cycle time, the 2-phase controller exhibits the highest performance, whereas the 4-phase controller shows the slowest performance. As in the case of the linear controller, this result supports 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 CB controllers were derived in equations (58), (63), and (68). Given that the gate delays are equal, we have roughly the same latencies for our controller and for the 2-phase controller, whereas the 4-phase controller has slightly lower latency. The minimum latencies obtained for each type of controller (equations (60), (65) and (70)) when there are no logic processing units in between stages also conform to the same latency order with the 4-phase controller being the lowest and our controller being the highest.

5. Implementation and Results

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

5.1 Implementation

To prove the concept, we evaluated the performance of each of the controllers on a 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 were placed identically in each case using the *rloc* placement constraints of the Xilinx ISE tool. Synthesis options, both general and Xilinx-specific ones were tuned to suit asynchronous design synthesis; for example, the use of global and regional clock buffers was disabled. Thus, we believe that the results we obtained are comparable with each other with minimum uncertainty in the measurements.

For the linear controllers, we created simple 8-bit 4-stage FIFOs operated by controllers of each type. For the CB controllers, we built 8-bit Y-shaped pipelines with 4-stages where two stages were in the stem and two stages were branched out. The CB controller was placed in the second stage of the pipeline. Data width of all pipelines were 8-bit.

Environment of the pipelines comprised input-generating shift registers and output-capturing registers (two registers in the CB case) were operating with minimum overhead, which maximized 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 was minimum delay between stages without any logic processing in between which represents the maximum performance of the controllers for high-speed pipelines. Since there was no logic processing ($t_{logic} = 0$), no matched delays were inserted between stages either ($t_{MD} = 0$).

In the second case, we tested the performance of pipeline controllers for a general scenario of pipelines operating with processing in between stages. To emulate the processing elements, we used simple buffers to delay the data path. The introduced logic delay ranged from 6.9ns to 7.2ns (varied depending on the exact routing of the data path) for each stage of the pipeline. This delay was chosen such that it satisfied the condition (55) (which in turn satisfies condition (27)) that we derived for CB controller for the EA protocol. Thus, we could obtain the performance of the EA protocol (and other protocols) in the case of a general pipeline with logic processing, for which these two conditions can be easily satisfied.

The delay elements for controllers were tuned starting from a higher delay to the lowest possible delay for which proper operation of the pipeline was guaranteed. Even when the logic delay was measured accurately, it was not possible to get the exact delay value. The tracking error, i.e., the difference between the actual and required delays, should be always kept positive to correctly operate a design. This error is additive to the cycle time of each design and is common to all designs of the three protocols. In an ASIC design, adjustable delays are desired so that the delay can be set properly after the design has been fabricated.

5.2 Results

Post-layout simulation results for Vertex-4 obtained using ModelSim are shown in Table 1. The first column of results shows, that the 2-phase controllers outperformed the 4-phase and EA controllers in linear and CB operations when there was no processing in between pipeline stages ($t_{logic} = 0$). Its performance advantage was evident in these cases where minimum controller overhead is desirable. Since $t_{logic} = 0$, condition (27) for the EA controller does not hold, so the overhead of the controller is exposed on the critical cycle time which explains its larger cycle time.

The second column shows that, in the cases where logic processing is present between pipeline stages, the EA controllers performed better as their overhead got hidden in the required delay between stages. For EA controllers, condition (27) holds in this case, so their performances are comparable to that of the 2-phase controller in linear operation, which confirms the analytical cycle times that we obtained in (28) and (33).

The last thee rows of the second column show, that the CB controller for EA protocol outperformed the 4-phase controller and performed slightly better than the 2-phase controller. As we demonstrated in our analysis, the ability of the EA protocol to hide the control overhead leads to this performance gain. In our FPGA implementation, all gates (including the C-elements) are implemented using lookup tables (LUTs) that have identical delays, which simplifies the comparison of controller cycle times. Given the equal delays in gates, the difference in cycle times of CB controllers derived in (56) and (62) amounts to a 2-gate delay, which is roughly between 600 ps and 1100 ps [19] in the Vertex-4 architecture. Hence, the results (600 ps difference) agree with our formal analysis, subject to routing delay variations. Even though the gain is relatively small, the simplicity of the EA protocol as a 4-phase protocol makes it more appealing in this case, which is a non linear asynchronous pipeline application. As described earlier, when the 2-phase protocol is used, translations from 2-phase protocol to 4-phase protocol are usually required at some points where level-sensitive control is necessary. In such cases, our controller has the added advantage using a variation of the 4-phase protocol and of having a performance gain over the 2-phase protocol by hiding the additional controller overhead incurred by non linear operations.

Table 1 Cycle-time comparison.

Cycle Time (ns)	Without processing	With processing
Linear		
2-phase (MOUSETRAP)	2.6	9.9
4-phase	3.6	12.4
EA	4.0	10.0
CB		
2-phase	4.0	11.2
4-phase	7.1	15.1
EA	5.6	10.6

To evaluate the area consumption of our controller, we measured the resource utilization of the FPGA for our designs. The resource utilization of the control path for *both* controllers and matched delays in terms of FPGA slices used in the 4-stage pipelines used in our experiments is shown in Table 2. In the Vertex-4 architecture that we used, one slice comprises two lookup tables and two flipflops and/or latch units.

Table 2 Resource utilization comparison.

Resource Utilization (of slices)	Without processing	With processing
Linear		
2-phase	4	24
4-phase	4	27
EA	12	26
CB		
2-phase	8	43
4-phase	8	41
EA	20	42

The first column of Table 2 shows the resource utilization of pipelines without processing for linear and CB controllers for all three protocols. Since we did not use any matched delays in this case, this column shows the resources consumed by controllers alone. In the linear 4-stage FIFO, both 2-phase and 4-phase controllers have the same resource utilization; 4 slices. EA controllers, which consume the most resources, utilize 12 slices in total for their logic and for the self-resetting delay elements inside. However, when we consider the case with logic processing (second column), where the control path now includes both control logic and matched delays, the resource usage of EA controllers (26 slices) is comparable to that of the 2- and 4-phase controllers (24 and 27 slices, respectively). The reason for this is that EA controllers require a *smaller* matched delay $t_{MD\uparrow}$ for a logic processing stage given the same t_{logic} compared with the other two protocols.

The same reasoning applies to the case of CB controllers. Even though EA controllers themselves consume higher resources, the total resources consumed in the case of a pipeline with processing is comparable in all thee cases. Hence, we could obtain the performance gains described earlier with more or less the same resource utilization for the control path, which highlights the advantages of using EA protocol.

6. Conclusions and Future Work

We proposed a new pipeline controller for the Early Acknowledgement (EA) 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 found both analytically and experimentally that the controller's cycle-time was comparable to 2-phase controller MOUSETRAP.

Furthermore, we highlighted on the advantages of using EA protocol, which inherits the simplicity of the 4-phase protocol, by comparing the conditional branch controllers for each protocol. The area usage of the protocol is also comparable to those of other two protocols in the preferred application of this protocol since the required matched delay is smaller, so that the control path (controllers and matched delays together) do not have a area penalty.

We would like to evaluate and confirm the performance of the controllers on an ASIC, such as 65-nm technology. With think that experimental results in such a case are necessary to strengthen our claims for the advantages of using the EA protocol.

7. Acknowledgements

This work is supported in part by The Graduate University for Advanced Studies in Japan.

References

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

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

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

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

[5] M. Singh, S.M. Nowick, *MOUSETRAP: High-Speed Transition-Signaling Asynchronous Pipelines*, IEEE Transactions on VLSI Systems, pages 684-698, 2007.

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

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

[8] E. Brunvand, *Using FPGAs to Implement Self-Timed Systems*, Journal of VLSI Signal Processing, Vol. 6, Issue 2, pages 173-190, August 1993.

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

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

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

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

[13] Y. Sato, et al. *Systematic Reducing of Metastable Operation Occurred in CMOS D Flip-Flops*, Systems and Computers in Japan, pages 1090-1098, 2000.

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

[15] K.V. Berkel, F. Huberts and A. Peeters, *Streching Quasi Delay Insensitivity by Means of Extended Isochronic Forks*, Proceedings on Asynchronous Design Methodologies, pages 99-106, 1995.

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

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

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

[19] ALTERA (White Paper) *Stratix II vs. Virtex-4 Performance Comparison*

[20] J. Bengtsson, et el. *Uppal — a Tool Suite for Automatic Verification of Real–Time Systems*, Proceedings of Workshop on Verification and Control of Hybrid Systems III, pages 232–243, 1995.

[21] UPPAL Model Checker, http://www.uppaal.com/

[22] Private communication with Montek Singh

Chammika Mannakkara received a BSc. in Electrical and Electronic Engineering from the Faculty of Engineering, University of Peradeniya, Sri Lanka in 2000. He joined the 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 the National Institute of Informatics, Tokyo, supervised 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 the National Institute of Informatics in 2002, where he is currently a Professor. He was a visiting researcher at Carnegie Mellon University from 1990 to 1991. His research activities are currently focused on formal verification of hardware. Dr. Yoneda is a member of IEEE, the Institute of Electronics, Information, and Communication Engineers of Japan, and the Information Processing Society of Japan.

Appendix A: Performance analysis of 2-phase linear controller

Here we use the STG diagram presented in [4] (with our naming conventions for the control signals. See Fig. A· 1) to derive the performance of 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" except for the *enable* signals. 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. 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,

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

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: $Ain_N \rightarrow Rin_{N+1} \rightarrow Aout_N \rightarrow enable_{N+1}-$

$$T_1 = t_{MD}^{N+1} + t_{latch} + t_{XNOR\downarrow}. \qquad \text{(A· 3)}$$

- Path on data cycle: $Ain_N \rightarrow enable_N- \rightarrow enable_{N+1}-$

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

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

$$\begin{aligned} t_{MD}^{N+1} + t_{latch} + t_{XNOR\downarrow} &\geq t_{XNOR\downarrow} + t_{logic} + t_{latch} \\ t_{MD}^{N+1} &\geq t_{logic}. \qquad \text{(A· 5)} \end{aligned}$$

In other words, the matched delay should be selected to be equal or greater than the logic delay. The optimal matched delay is when $t_{MD}^{N+1} = t_{logic}$. The cycle time and forward latency will be:

$$\begin{aligned} T_{2P}^l &= t_{logic} + 2 \cdot t_{latch} + t_{XNOR\uparrow} \qquad \text{(A· 6)} \\ L_{2P}^l &= t_{logic} + t_{latch}. \qquad \text{(A· 7)} \end{aligned}$$

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:

$$\begin{aligned} T_{2P}^l|_{min} &= 2 \cdot t_{latch} + t_{XNOR\uparrow} \qquad \text{(A· 8)} \\ L_{2P}^l|_{min} &= t_{latch}. \qquad \text{(A· 9)} \end{aligned}$$

Appendix B: Performance analysis of 4-phase linear controller

The STG diagram 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.

$$\begin{aligned} T &= t_{G1\uparrow} + t_{G2\uparrow} + t_{MD\uparrow}^{N+1} + t_{G1\uparrow} \\ &\quad + t_{G2\downarrow} + t_{MD\downarrow}^{N+1} + t_{G1\downarrow} \qquad \text{(A· 10)} \\ L &= t_{G1\uparrow} + t_{G2\uparrow} + t_{MD\uparrow}^{N+1} + t_{G1\uparrow}. \qquad \text{(A· 11)} \end{aligned}$$

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

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

$$T_1 = t_{G1\uparrow} + t_{G2\uparrow} + t_{MD\uparrow}^{N+1} + t_{G1\uparrow}. \qquad \text{(A· 12)}$$

- 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:

$$t_{MD\uparrow}^{N+1} \geq t_{flop} + t_{logic} - (t_{G1\uparrow} + t_{G2\uparrow})$$

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

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

$$\begin{aligned} T_{4P}^l &= t_{flop} + t_{logic} + t_{G1\uparrow} + t_{G1\downarrow} \\ &\quad + t_{G2\downarrow} + t_{AND\downarrow} \qquad \text{(A· 15)} \\ L_{4P}^l &= t_{flop} + t_{logic} + t_{G1\uparrow}. \qquad \text{(A· 16)} \end{aligned}$$

(The $t_{MD\downarrow}^{N+1}$ has been replaced with equivalent $t_{AND\downarrow}$.) When the above condition does not hold the above parameters can be deduced from equations (A· 10) and (A· 11) at $t_{MD\uparrow} = t_{MD\downarrow} = 0$.

$$\begin{aligned} T_{4P}^l|_{min} &= 2 \cdot t_{G1\uparrow} + t_{G2\uparrow} + t_{G1\downarrow} + t_{G2\downarrow} \qquad \text{(A· 17)} \\ L_{4P}^l|_{min} &= 2 \cdot t_{G1\uparrow} + t_{G2\uparrow}. \qquad \text{(A· 18)} \end{aligned}$$

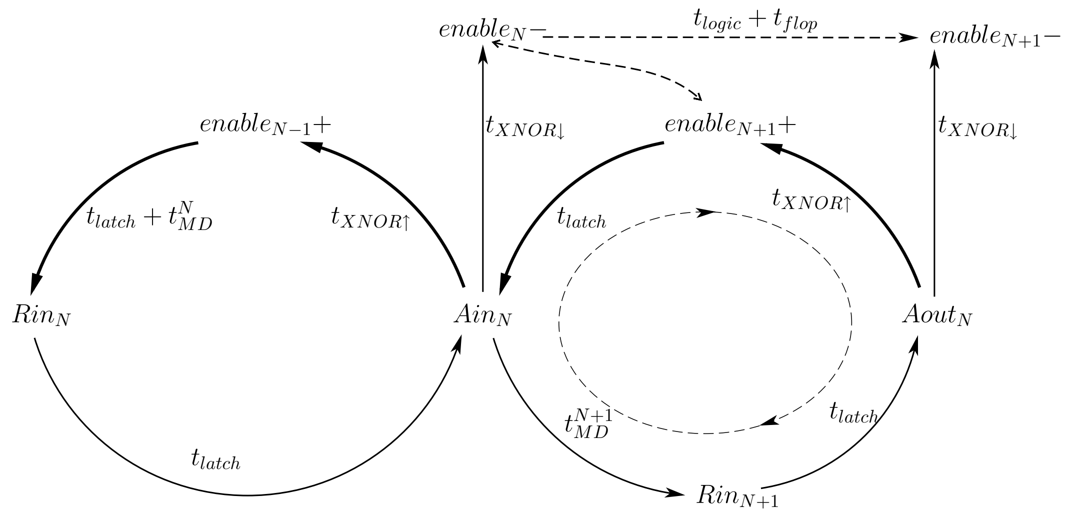
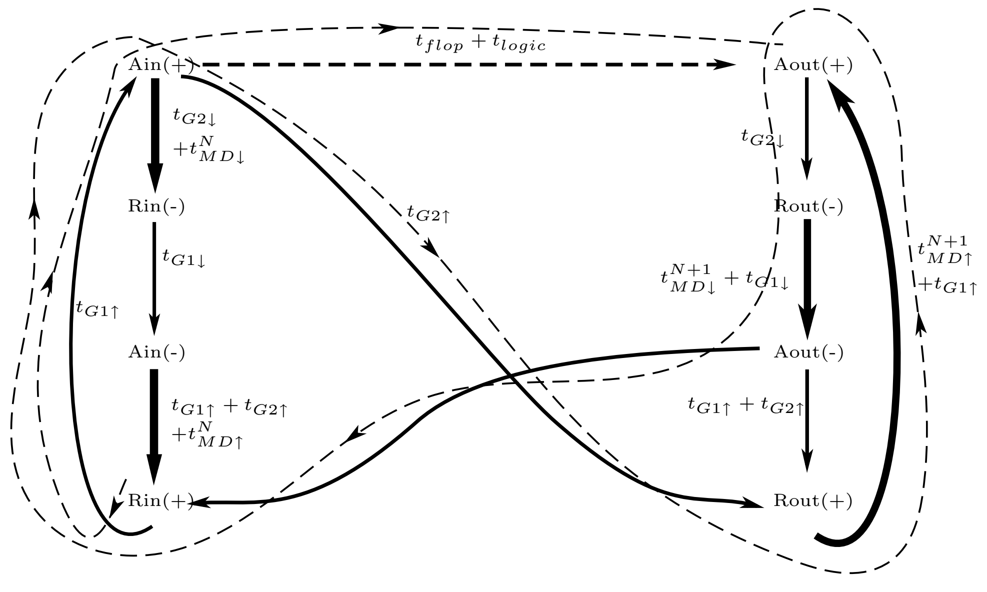


Fig. A· 1 STG for MOUSETRAP Controller.



$t_{flop}+t_{logic}$
Ain(+)
Aout(+)
$t_{G2\downarrow}$ $+t_{MD\downarrow}^N$
$t_{G2\downarrow}$
Rin(-)
$t_{G2\uparrow}$
Rout(-)
$t_{G1\downarrow}$
$t_{MD\uparrow}^{N+1}$ $+t_{G1\uparrow}$
$t_{G1\uparrow}$
$t_{MD\downarrow}^{N+1}+t_{G1\downarrow}$
Ain(-)
Aout(-)
$t_{G1\uparrow}+t_{G2\uparrow}$ $+t_{MD\uparrow}^N$
$t_{G1\uparrow}+t_{G2\uparrow}$
Rin(+)
Rout(+)

Fig. A· 2 STG for Four phase Controller.

Appendix C: Performance analysis of CB controller for 2-phase protocol

The STG diagram for the functional operation of the 2-phase CB controller is given in Fig. A· 3. In this also, transitions for only one branch of the controller is given. Similar to 2-phase linear (MOUSETRAP) controller, the signals Rin, Ain, $Rout1$ and $Aout1$ in the STG diagram implies a "transition" without explicitly unfolding the particular transitions (from low-to-high and high-to-low) which are identical in the protocol. Moreover, the matched delays are symmetrical for the transition signalling protocol.
We can calculate the cycle time and forward latency in terms of t_{logic} using the same rationale used in CB controller of the EA protocol. The cycle time and forward latency of the controller as shown in the STG diagram with thin dashed lines, can be expressed as follows.

$$T = t_{flop} + t_{MD1} + t_{latch} + t_{XNOR\uparrow} + 2 \cdot t_{AND\uparrow} \quad \text{(A· 19)}$$

$$L = t_{XOR\uparrow} + t_{AND\uparrow} + t_{flop} + t_{MD1} + t_{latch} + t_{XNOR\downarrow}. \quad \text{(A· 20)}$$

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

- Path on control cycle: $clk_N+ \rightarrow Rout1 \rightarrow Aout1 \rightarrow$

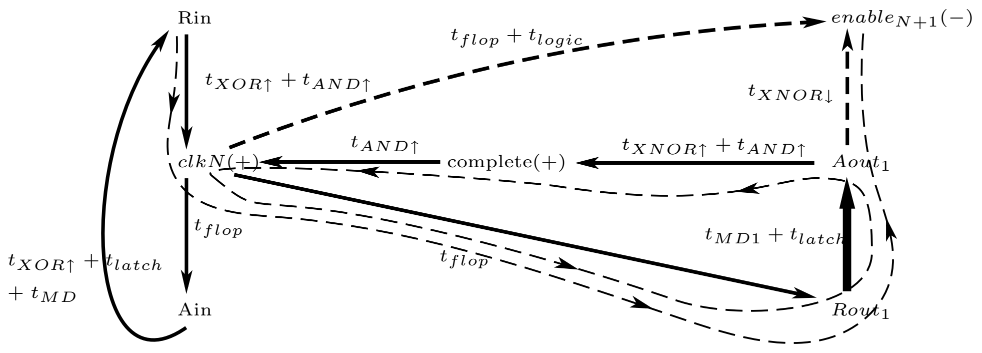


Fig. A· 3 STG for CB controller for 2-phase protocol.

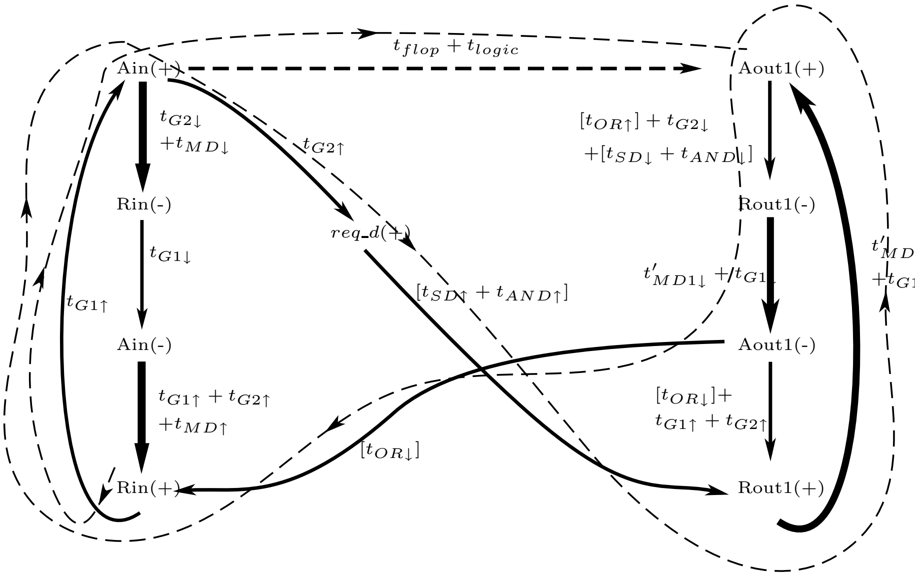


Fig. A· 4 STG for CB controller for 4-phase protocol.

$enable_{N+1}-$

$$T_{CB1} = t_{flop} + t_{MD1} + t_{latch} + t_{XNOR\downarrow}. \quad \text{(A· 21)}$$

- Path on data cycle: $clk_N+ \rightarrow enable_{N+1}-$

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

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

$$t_{MD1} \geq t_{logic} - (t_{latch} + t_{XNOR\downarrow}) \quad \text{(A· 23)}$$

thus, if, $\quad t_{logic} \geq t_{latch} + t_{XNOR\downarrow} \quad$ (A· 24)

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

$$T^{cb}_{2P} = t_{flop} + t_{logic} + 2 \cdot t_{AND\uparrow} + (t_{XNOR\uparrow} - t_{XNOR\downarrow}) \quad \text{(A· 25)}$$

$$L^{cb}_{2P} = t_{flop} + t_{logic} + t_{XOR\uparrow} + t_{AND\uparrow}. \quad \text{(A· 26)}$$

When the above condition does not hold we can derive the cycle time and latency directly from (A· 19) and (A· 20) at $t_{MD1} = 0$ as follows.

$$T^{cb}_{2P}|_{min} = t_{flop} + t_{latch} + t_{XNOR\uparrow} + 2 \cdot t_{AND\uparrow} \quad \text{(A· 27)}$$

$$L^{cb}_{2P}|_{min} = t_{XOR\uparrow} + t_{AND\uparrow} + t_{flop} + t_{latch} + t_{XNOR\downarrow}. \quad \text{(A· 28)}$$

Appendix D: Performance analysis of CB controller for 4-phase protocol

The Fig. A· 4 shows the STG diagram for the CB controller for 4-phase protocol. The analysis of the cycle time and forward latency is similar to that of EA protocol CB 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 CB controller for EA 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 diagram, can be expressed in terms of gate delays as follows.

$$\begin{aligned} T &= t_{G1\uparrow} + t_{G2\uparrow} + [t_{SD\uparrow} + t_{AND\uparrow}] + t'_{MD1\uparrow} \\ &\quad + t_{G1\uparrow} + [t_{OR\uparrow}] + t_{G2\downarrow} + [t_{SD\downarrow} + t_{AND\downarrow}] \\ &\quad + t'_{MD1\downarrow} + t_{G1\downarrow} + [t_{OR\downarrow}] \qquad \text{(A$\cdot$ 29)} \\ L &= t_{G1\uparrow} + t_{G2\uparrow} + [t_{SD\uparrow} + t_{AND\uparrow}] \\ &\quad + t'_{MD1\uparrow} + t_{G1\uparrow}. \qquad \text{(A$\cdot$ 30)} \end{aligned}$$

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: $Rin+ \rightarrow Ain+ \rightarrow req\_d+ \rightarrow Rout1+ \rightarrow Aout1+$

$$\begin{aligned} T_1 &= t_{G1\uparrow} + t_{G2\uparrow} + [t_{SD\uparrow} + t_{AND\uparrow}] \\ &\quad + t'_{MD\uparrow} + t_{G1\uparrow}. \qquad \text{(A$\cdot$ 31)} \end{aligned}$$

- Path on data cycle: $Rin+ \rightarrow Ain+ \rightarrow Aout1+$

$$T_2 = t_{G1\uparrow} + t_{flop} + t_{logic}. \qquad \text{(A$\cdot$ 32)}$$

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

$$\begin{aligned} t'_{MD\uparrow} &\geq t_{flop} + t_{logic} - (t_{G1\uparrow} + t_{G2\uparrow} \\ &\quad + [t_{SD\uparrow} + t_{AND\uparrow}]) \\ \text{Thus, if,} \quad t_{logic} &\geq (t_{G1\uparrow} + t_{G2\uparrow} + [t_{SD\uparrow} + t_{AND\uparrow}]) \\ &\quad - t_{flop} \qquad \text{(A$\cdot$ 33)} \end{aligned}$$

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

$$\begin{aligned} T_{4P}^{cb} &= t_{flop} + t_{logic} + t_{G1\uparrow} + t_{G1\downarrow} + t_{G2\downarrow} \\ &\quad + t_{AND\downarrow} + [2 \cdot t_{AND\downarrow} + t_{OR\uparrow} + t_{OR\downarrow}] \qquad \text{(A$\cdot$ 34)} \\ L_{4P}^{cb} &= t_{flop} + t_{logic} + t_{G1\uparrow}. \qquad \text{(A$\cdot$ 35)} \end{aligned}$$

($t_{SD\downarrow}$ has been replaced with equivalent delay $t_{AND\downarrow}$.) When the above condition does not hold the minimum values for above parameters can be deduced from equations (A$\cdot$ 29) and (A$\cdot$ 30) at $t'_{MD\uparrow} = t'_{MD\downarrow} = 0$.

$$\begin{aligned} T_{4P}^{cb}|_{min} &= 2 \cdot t_{G1\uparrow} + t_{G2\uparrow} + t_{G1\downarrow} + t_{G2\downarrow} \\ &\quad + [t_{SD\uparrow} + t_{AND\uparrow} + 2.t_{AND\downarrow} \\ &\quad + t_{OR\uparrow} + t_{OR\downarrow}] \qquad \text{(A$\cdot$ 36)} \\ L_{4P}^{cb}|_{min} &= 2 \cdot t_{G1\uparrow} + t_{G2\uparrow} + [t_{SD\uparrow} + t_{AND\uparrow}]. \\ &\qquad \text{(A$\cdot$ 37)} \end{aligned}$$