Low latency synchronization through speculation

D.J.Kinniment, and A.V.Yakovlev

School of Electrical and Electronic and Computer Engineering, University of Newcastle, NE1 7RU, UK
{David.Kinniment,Alex.Yakovlev}@ncl.ac.uk

Abstract. Synchronization between independently clocked regions in a high performance system is often subject to latencies of more than one clock cycle. We show how the latency can be reduced significantly, typically to half the number of clock cycles required for high reliability, by speculating that a long synchronization time is not required. The small number of synchronization metastability times longer than normal are detected, and subsequent computations re-evaluated, thus maintaining the reliability of the system.

1. Introduction

Systems on silicon are increasingly designed using IP blocks from different sources. These blocks are optimised to operate at a particular clock rate and there may therefore need to be several different clocks in the system. If the clocks are all phase locked, there is no need for re-synchronisation of data passing between from block to another, since data originating in one clock domain and passing to the next will always arrive at the same point in the receiving clock cycle. Even if the phase difference is not the same, it may not be known until the clock has run for some time, but it is then predictable, and a suitable data exchange time can be found, [1].
In practice it is difficult to achieve accurate and reliable locking between all the clock domains for a number of reasons.

- Jitter and noise in the clock generation circuit, particularly if two different frequencies must be locked.
- Crosstalk between the data and the clock tree connections introducing noise into both.
- Dynamic power supply variations between different parts of a clock tree and the data distribution, affecting the path delay difference.
- Temperature changes may alter the relative phase delay of two clock trees designed in different ways.
- Input output interfaces between the system and the outside world are not controllable and phase relationships cannot be predicted.

These effects cause unpredictable variation in the time of arrival of a data item relative to the receiving clock, which is particularly noticeable in high performance systems using IP blocks with large clock trees [2]. Figures of 150ps noise [4]and 110ps clock skew [3], have been reported in 0.18μ systems, and this is likely to increase with smaller geometries. Design of interfaces in high performance systems with fast clocks and large timing uncertainties becomes more difficult as these uncertainties increase as a proportion of the receiving clock cycle and it may be simpler to assume that the two domains are independent.
Exchange of data between two independently clocked domains requires the use of a synchronizer. This ensures that data originally synchronized to the writer's clock is synchronized to the reader's clock so that it arrives at the correct time in relation to the reader's computation cycle. Conventionally this is done with the circuit of Fig. 1, [5].

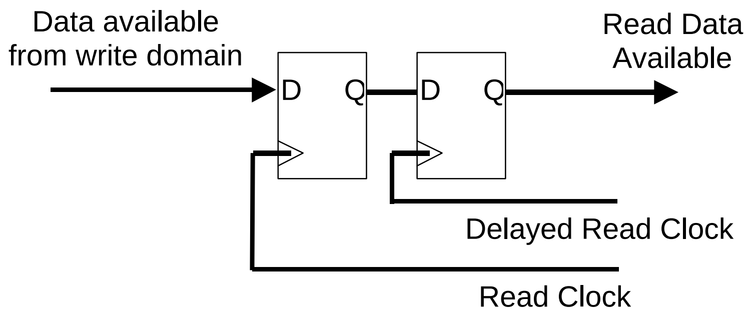

Fig. 1. Synchronizer

Here metastability may occur in the first flip-flop, caused by the asynchronous nature of the write data available signal and the read clock. This is allowed to settle for a fixed amount of time before being clocked into the second flip flop by the later delayed read clock with the aim of obtaining a clean read data available signal. The reliability of the synchronizer depends critically on the time allowed between the read clock and the delayed read clock, which determines the time allowed for metastability to settle. In terms of the metastability time constant, t, of the flip-flop, increasing the delay by a time t increases the MTBF by a factor of $e^{\frac{t}{t}}$, and in order to keep system failures to less than 1 in 10^8 s (2 years) the total settling time may need to be over $30t$. Since t is comparable to a gate delay, $30t$, may be equivalent to several clock cycles, if t = 50ps then a single clock cycle at 1 GHz is insufficient. Using several clock cycles for synchronization can lead to unacceptably large latencies because the data cannot be read until the read data available is asserted. While throughput can be increased by using an intermediate FIFO to separate the read and write operations, the total time between generation of the data and its availability in the receiver cannot. As a result, considerable latency is added to the data transmission between separately clocked zones, and this latency can be a problem, particularly in long paths where a computation cannot proceed until a requested result is received.
By allowing the read data to be used as soon as metastability is complete, fully asynchronous systems, or those with stoppable clocks allow the data transmission latency to be closer to the average metastability time rather than always using that required to accommodate the worst metastability time [6]. We will show that it is also possible to reduce the synchronization latency in a fully synchronous system, by assuming that the synchronization time does not normally need to be long, but allowing for recovery from any errors if an event with a long metastability time is encountered. In section 2 we describe how a FIFO is normally used the achieve high throughput, In section 3 how possible errors can be detected, and in section 4 we show the operation of a synchronizer circuit with error detection. Finally a simple system is described in section 5 where error detection and recovery is combined with a FIFO to give high throughput as well as lower than normal latency.

2. High throughput synchronization

Synchronization of the data available signal and any acknowledgement of the reading of the data may require a number of clock cycles, and therefore there can be a delay between the write data becoming available and the actual reading of the data, and also between the acknowledgement of the reading and the writing of the next data item. This leads to low throughput, but inserting a FIFO between the writer and the reader as shown in Fig. 2, [7], allows both reading and writing to take place immediately if the FIFO is always partially filled, and always has some space available for a new write. These conditions are indicated by the two Flags Full, which is true if there is only sufficient space to accommodate a certain number of data items and Not Empty, which is true if there are at least a certain number of unread items in the FIFO. Provided there are always at least n data items in the FIFO, the delay of n cycles required to synchronize the Not Empty signal can be overlapped with the reading of these n items. Similarly with n spaces the writes can also be overlapped. Unfortunately this does not improve the latency, because the time between writing an item, and reading it is at least n cycles of the read clock plus between 0 and 1 cycle for the Not empty signal to appear as a Request. Similarly between n and n + 1 cycles of the write clock are needed to recognise that the FIFO needs another item. Forward latency can only be reduced by reducing the time needed to synchronize the Not Empty signals, and reverse latency by reducing the Full synchronization time.

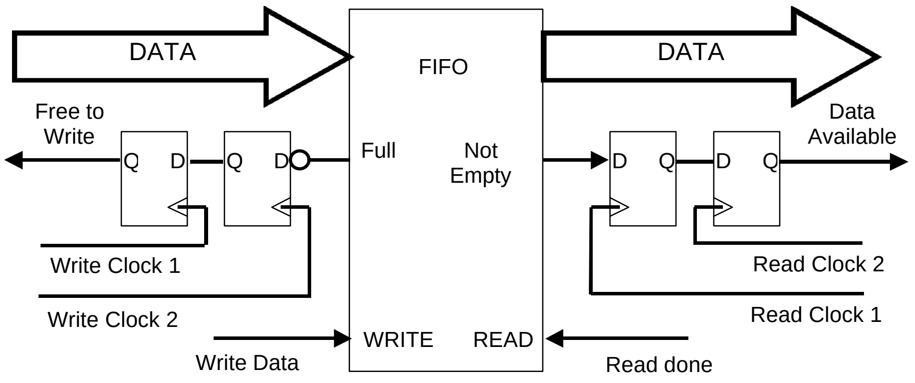

Fig. 2. Synchronizer with FIFO

3. Synchronization error detection

If the synchronization delay between the read clock and the delayed read clock is reduced the system performance is improved, but the probability of failure for each data transmission will be much higher, for example for a 10τ delay it would be 1 in 20,000. However if it were possible to later detect the cases where synchronisation failed, and recover by returning to the system state before failure, the performance of the system could be improved with little penalty in reliability. Note that this proposal does not eliminate the metastability problem, because there will still be cases where metastability lasts longer than the failure detector allows, and the system state will not be recoverable.

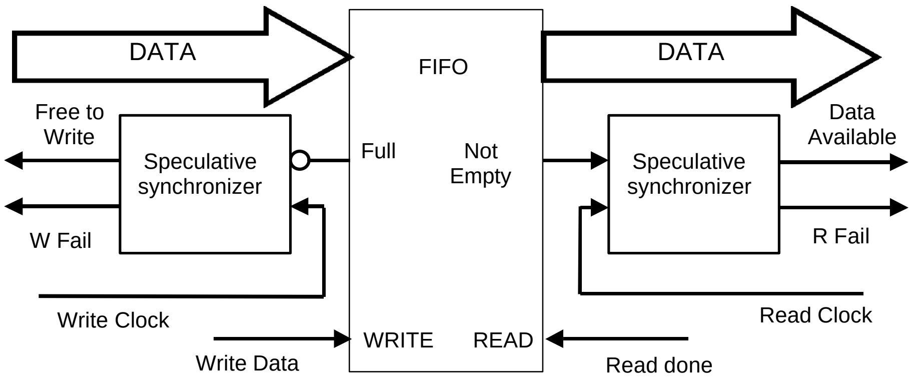

Fig. 3. Low latency synchronization

Fig. 3 shows how this can be done by replacing both conventional n-cycle synchronizers with speculative versions in which the data available, or Free to write signals are produced early, and allowed to proceed if they subsequently prove to be correct. However, if there is any doubt about their validity at a later time, the R Fail or W Fail flag is raise so that computation can be repeated.
In the speculative synchronizer the first flip-flop must be tested to detect whether it has resolved in a short time or whether it takes longer than average. In order to achieve this we must use a metastability filter on its output as shown in Fig. 4. This circuit, sometimes known as a Q-Flop [8], ensures that half levels will not appear at the output, and the uncertainty due to metastability appears (in the case of the read synchronizer) as a variable delay time from the synchronized write data available to the read data available output.

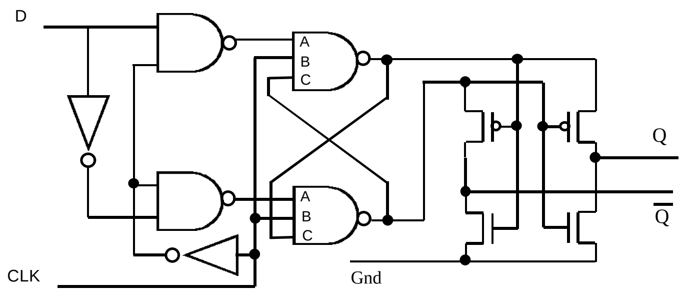

Fig. 4. Q-Flop

In the Q-Flop only clean level transitions occur, because the output is low during metastability, and only goes high when there is a significant voltage difference between Q and Q'. It can be shown that the transition between a voltage difference of V_1 and V_2 takes a time $t.\ln\frac{V_2}{V_1}$, so when the voltage between Q and Q' is sufficient to produce an output (more than the p-type threshold, V_t) the transition to a full output of V_{dd} will take less than $2t$. This voltage difference is also much greater than the flip flop internal noise level, so when the read data available signal leaves the metastable region to make a high transition, it happens in a fixed time and cannot return low. On the other hand, the time at which this transition takes place cannot be determined, and may be unbounded.

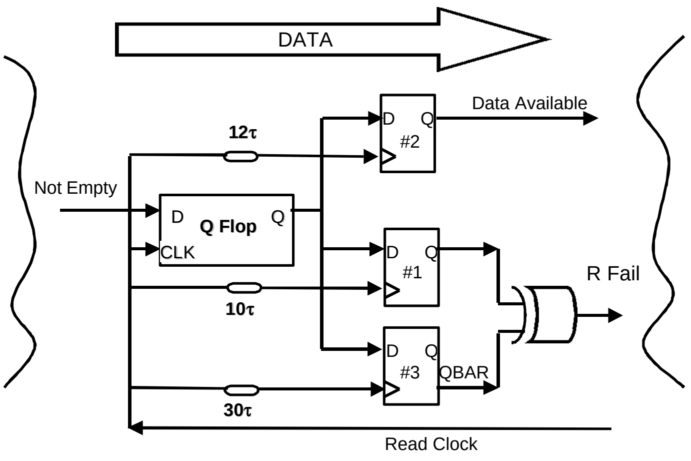

Fig. 5. Speculative synchronizer

If we use further timing signals derived from the read clock to sample the read data available at $10t$, $12t$, and $30t$, as shown in Fig. 5, it is possible to determine whether the partially synchronized read signal changed between $10t$, and $30t$.

We will assume that at $30t$ all the samples are stable, an assumption that is true to a high degree of probability. There are then only four combinations of the three samples to consider since if the $10t$ sample has reached a high level, the $12t$, and $30t$ samples must also be high. Similarly if the $12t$ sample has reached a high level, the $30t$ sample must also be high. These four combinations, together with the possibility of metastability at $30t$ in **Table 1**.

10τ	**12τ**	**30τ**	**Fail**	**Comment**
0	0	Metastable	?	Unrecoverable error
0	0	0	0	No data available
0	0	1	1	Change between 10τ and 30τ, return to original state
0	1	1	1	Change between 10τ and 30τ, return to original state
1	1	1	0	Normal data Transfer

Table 1. Synchronizer possibilities

The probability of the 30t sample being metastable, is e^{-30} and will be discounted. If the 10t, and 30t samples are both 0, we can infer that there was no data available signal, and the output at 12t was correctly flagged as 0. If the 10t, and 30t samples are both 1, we can infer that there was a data available signal, and the output at 12t was correctly flagged as 1. The only other possible case is that the 10t, and 30t samples are 0 and 1 respectively. In this case the output of the Q Flop changed some time between 10t, and 30t and the 12t value is unreliable. The fail signal is then set and the data transfer must be repeated. The MTBF for this arrangement is the same as that of a 30t synchronizer, because even if the 10t sample is metastable, the chances of this are 1 in e^{10}, and further20t is allowed for the metastability to settle. Given that it is metastable, chance of it remaining metastable at 30t is 1 in e^{20}, and the overall probability is 1 in $e^{10}.e^{20} = e^{30}$

4. Circuit operation

To verify the operation of the circuit of Fig. 5 we simulated it using SPICE to model the TSMC 0.18µ process. Trial and error simulations of the Q-flop constructed of standard gates as shown in Fig. 4 gave a τ of about 53ps, and typical delay of 280ps in the circuit. With the D input 1.97ps ahead of the Clock input the metastability time became longer than 12 τ, and with the clock data overlap increased to 2.15 ps , the delay was 600ps (280ps delay + 6τ). With a clock period of 1ns, the probability of getting an overlap within 0.18 ps of the metastable point would be less than once in 5000 events.
Fig 6 shows the clock to the Not Empty Q-flop is on the bottom plot at 100ps, and two delayed version of this clock., one 540ps after the clock (T14, derived from a chain of 14 inverters) and one at 630ps, (T16 from 16 inverters). The output from the Q-flop goes high at around 600ps (middle trace), so the synchronised Data Available misses the first delayed clock, but is high for the second. The final output is shown on the top trace and goes high at around 1ns. This is the case represented by the penultimate line in Table 1, where a Fail signal would appear at 30τ after the clock, a delay of about 1.9ns, since the 540ps sample is low, and the 1.5ns sample is high. Latency is about 900ps for the Data Available signal. For a standard synchronizer with 30τ reliability the latency would be 280 + 30x53 = 1870ps.

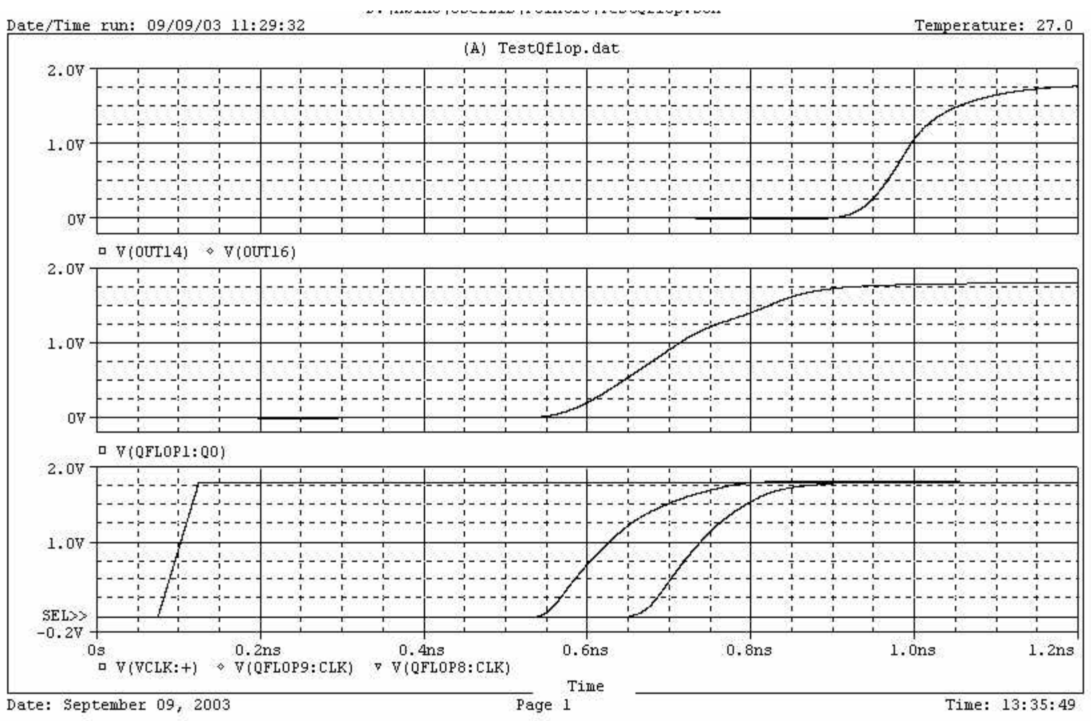

Fig. 6. Standard gate implementation

A value for τ of 53ps is significantly greater than that achievable with a 0.18μ, but it is possible to produce a Q-flop with a τ comparable with state or the art designs. We redesigned the Qflop by reducing the input coupling to the flip flop formed by the three input NAND gates of Fig 4. In the circuit of Fig. 7 just two small p types transistors are used to provide the A input, and some of the other transistors are resized to optimise the feedback from the C input. If B and C were both high at the same time that A was low, there could be a short circuit, but that cannot happen, except transiently, in the Q-flop because both A's must be high when the B inputs are high. This modification gives a τ of about 26ps (half the previous value). The times for each of the delayed clocks can now be reduced, and we use 10 inverters (379ps from the clock) and 12 inverters (491ps from the clock).

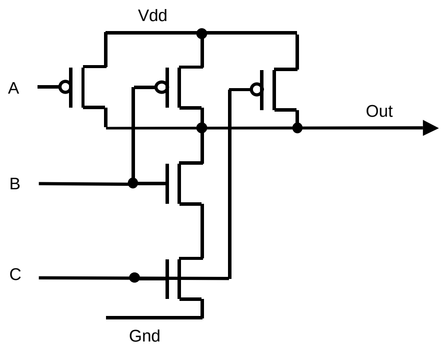

Fig. 7. Modified 3 input NAND gate

This is shown in the SPICE plot of Fig. 8. A Q-flop output response at 511ps (411ps from the clock) is again chosen to illustrate the Fail situation. Responses less than 379ps fail, and this occurs in about 1 in 5000 cases, since the 10 inverter point is 379ps after the clock, and delay in the Qflop is about 200ps leaving 179ps = 7τ approx. Total latency is about 650ps for the Data Available signal, compared with 200 + 30x26 = 980ps in a conventional synchronizer

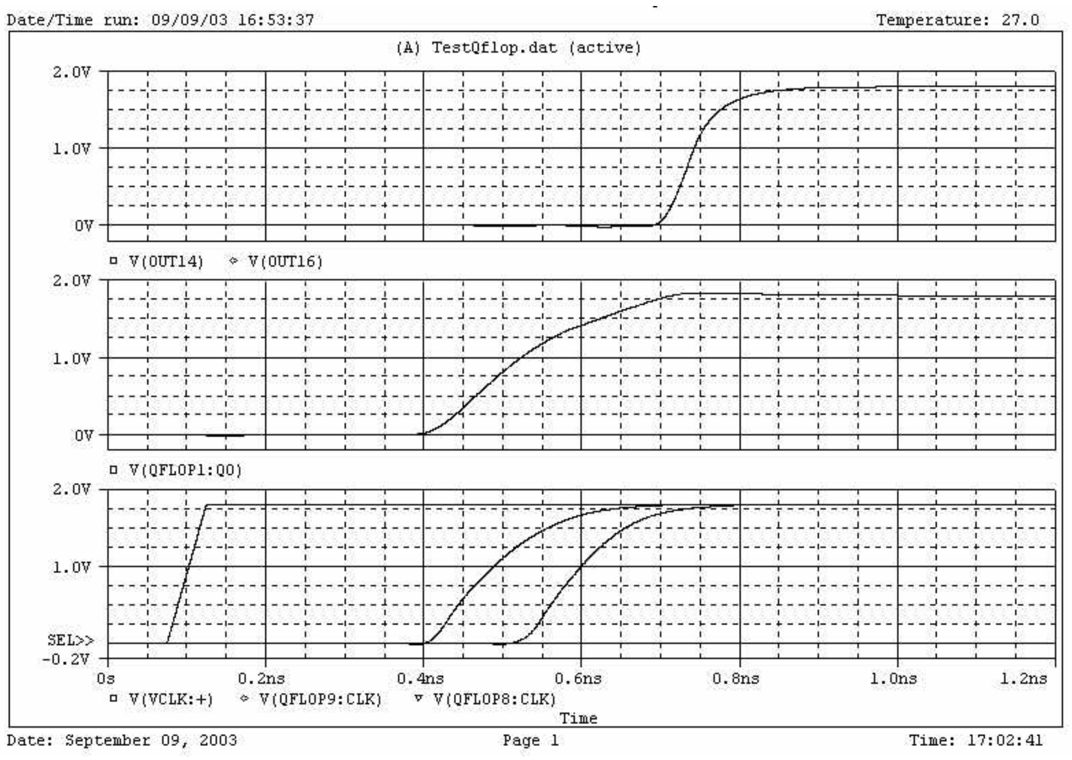

Fig. 8. Synchronizer with modified gate

5. Recovery

A clock period of 20τ would allow synchronization latency of between 1 and 2 clock cycles, but the fail signal is not generated until midway in the second cycle, so it is necessary to duplicate the Q-flop so that metastable events created in one cycle are preserved until the second. This is done with the arrangement of Fig. 9, where a data available is generated within one cycle, but the fail signal corresponding to that Data Available appears in the following cycle.

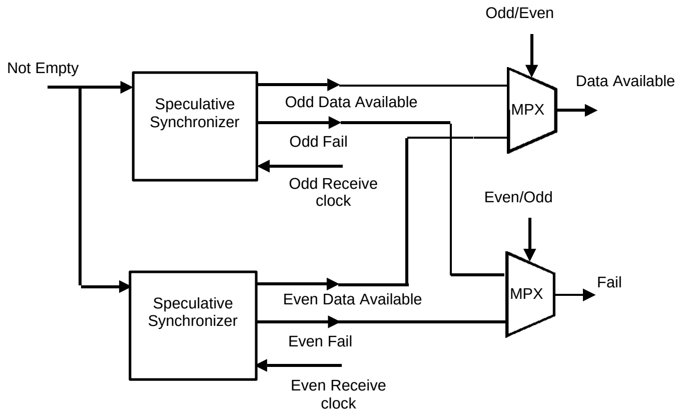

Fig. 9. Pipelined speculative synchronizer

To show how recovery can be achieved after a synchronization failure, we give a simple example in Fig. 10. Here input data from the writer is normally clocked into the input register, and then added to the R register which has a slave refreshed whenever the input register is overwritten. Read done is signalled back to the synchronizer during the next cycle by the flip-flop B which is clocked by a delayed read clock sufficiently far before the following cycle to enable the read pointer to be updated. We will assume that the receive clock period is 20τ, that the data available signal appears at 12τ in the first cycle, and that a fail signal does not arrive until middle of the second cycle at 30τ.

If there is a failure of synchronisation such that the data available signal is still metastable at the receive clock time, the input register, the slave, and the flip-flop A may all be corrupted. At 30τ the fail signal will force the flip flop B to be set to 0, and the R register master will not be altered. Since the read done is not generated, the read pointer is not updated, and the data available is effectively allowed another cycle to settle before both the input and R register slave are refreshed.

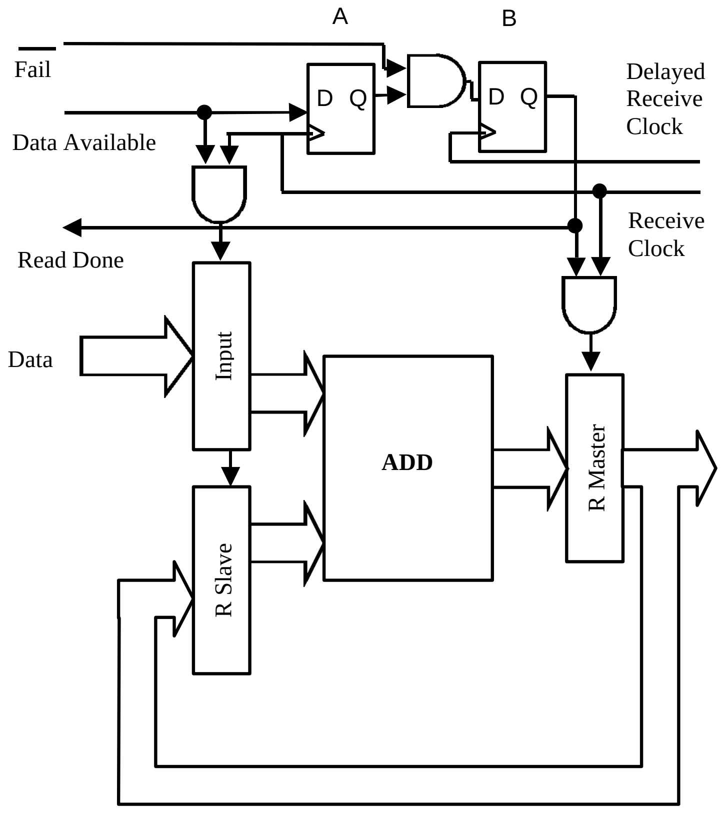

Fig. 10. Example of failure recovery

In this scheme only one extra cycle is required every time a synchronization failure occurs, and provided the rate of writing is slower than the rate of reading this extra delay will rapidly be made up. Average latency is now 0 – 1 read clock cycle for the time between movement of the write pointer which may alter the state of the Not Empty signal, and the next read clock cycle, plus 1 cycle for synchronisation plus 1 in e^{10} cycles for possible failures.

This is a relatively simple example, but in general, our strategy is to stop a pipeline when the failure is detected, and recompute or refresh corrupted values. It is important here to ensure that corrupted data, or control states are always recoverable, but this does not usually present a problem. This is similar to schemes in which a fast, but unreliable computation path may need to be backed up by a slower, but reliable one [9]. In more complex systems, or those in which failure cannot be detected in one cycle more than the read, every register that may be altered during a calculation dependent on correct input data is replaced by a stack. This stack needs to be of sufficient depth to cover the number of clock cycles between the initial read clock, and the final sample ($30t$), then the following strategy will ensure recovery:

a. Push each new value on to the head of each stack.
b. If failure occurs pop the stacks by a number sufficient to recover the old values, and repeat the transfer.

This procedure may take several clock cycles, but only need occur very infrequently.

6. Conclusion

Synchronization in systems where the relationship between write and read clock domains is unpredictable may require more than one clock cycle. This is particularly true of high performance system where the clock period is only a small number of gate delays. Normally the latency of a synchronizer is fixed at the time required to give an acceptable reliability. Here, if a synchronization time of n cycles is needed to achieve adequate reliability, say e^{-30}, for every event, then it will usually be possible to approximately halve the synchronization time while accepting a probability of e^{-10} (1 in 20,000) that a computation may have to be carried out again. In this case the latency is variable, but the worst case is no greater than that of the standard solution.
This recomputation procedure may take more than one clock cycle, but only need occur approximately once in several thousand data transmissions, so does not significantly impact on the system performance. Because the synchronization delay is reduced, data transmission latency is also be reduced by a factor of approximately two when compared to conventional; synchronizers. The additional hardware cost for the simple example described here is minimal, but latency is reduced from 2.5 cycles to approximately 1.5 cycles. In more complex examples, as the number of cycles required for synchronization in creases, as does both the hardware and the recovery time needed, and this must be traded off against the latency improvements.

7. References

1. A. Chakraborty, and M. Greenstreet, “Efficient Self-Timed Interfaces for crossing Clock Domains”. Proceedings ASYNC2003, Vancouver, 12 – 16 May 2003, pp78-88.
2. N.A Kurd, J.S. Barkatullah, R.O.Dizon, T.D.Fletcher, and P.D.Madland “Multi-GHz Clocking Schemes for Intel Pentium 4 Microprocessors” Proc. ISSCC 2001 Feb 2001 pp404-405
3. S Tam, S Rusu, U. N. Desai, R Kim, J. Zhang, and I Young., “Clock Generation and Distribution for the first IA-64 Microprocessor”. IEEE JSSC Vol. 35 No.11, Nov. 2000, pp1545-1552.
4. Y. Semiat, and R. Ginosar “Timing Measurements of Synchronization Circuits” Proceedings ASYNC2003, Vancouver, 12 – 16 May 2003, pp68-77.
5. C.Dike and E.Burton. “Miller and Noise Effects in a Synchronizing Flip-Flop”. IEEE Journal of Solid State Circuits Vol. 34 No. 6, pp.849-855, June 1999
6. S.W.Moore, G.S.Taylor, P.A.Cunningham, R.D.Mullins and P.Robinson. “Using Stoppable Clocks to Safely Interface Asynchronous and Synchronous sub-systems”. Proceedings AINT2000, Delft, 19-20 July 2000 pp.129-132.
7. W.J.Dally, and J.W.Poulton “Digital Systems Engineering” Cambridge University Press, 1998.
8. F.U.Rosenberger, C.E.Molnar, T.J.Chaney and T-P. Fang, “Q-Modules: Internally Clocked Delay-Insensitive Modules”. IEEE Transactions on Computers, Vol. 37, No. 9, Sept 1988, pp 1005-1018
9. A. Bystrov, D. Sokolov, and A. Yakovlev “Low Latency Control Structures with Slack” Proceedings ASYNC2003, Vancouver, 12 – 16 May 2003, pp164-173.