

Using Stoppable Clocks to Safely Interface Asynchronous and Synchronous Subsystems

S.W. Moore, G.S. Taylor, P.A. Cunningham, R.D. Mullins and P.Robinson
University of Cambridge, Computer Laboratory,
New Museums Site, Pembroke Street,
Cambridge CB2 3QG, U.K.
Simon.Moore@cl.cam.ac.uk

1. Introduction

This paper presents a low latency, high bandwidth and reliable interface between asynchronous and synchronous subsystems. The approach relies on stretching the clock to prevent metastability in the data flip-flops rather than simply waiting for the data flip-flops to resolve metastability. However, unlike existing clock stretching approaches, ours almost never needs to stretch the clock so the optimum clock frequency can be maintained.

We assume a bundled data constraint for communication between subsystems, i.e. for each data bus there is an associated request signal which indicates when the data is valid. Bundled data communication structures can provide a reliable asynchronous interconnect between independently clocked synchronous domains. This communications structure is in contrast to Q-modules [5] which sample input signals at regular intervals regardless of whether they are changing.

We also assume that the synchronous subsystems, for which we are providing a clock, are typically quite large with several stages of pipeline and often require a clock which is rarely stopped. This is in contrast to circuits which clock a single synchronous pipeline stage each time there is new input stimuli [2] which are simple extensions to micropipelines [9].

2. The simple two flip-flop synchroniser

Synchronising an asynchronous data stream is a well known problem which can be crudely resolved by latching the data at least twice to allow time for metastability in the flip-flops to resolve (Figure 1). This does not prevent metastability from propagating though the chance is small [4]. A more pressing concern is the latency that is introduced by this scheme. In the best case the asynchronous interface presents a request just before a clock edge and it then takes a further clock cycle for the data to appear, i.e.

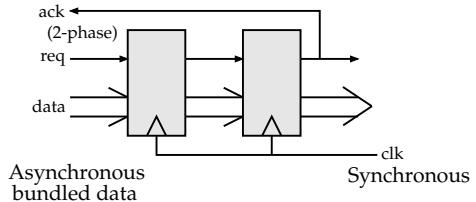


Figure 1. Two flip-flop synchroniser

just over one clock cycle in total. In the worst case it takes 2 clock cycles. The latency could be reduced by clocking the two sets of flip-flops from different edges of the clock. However, this would reduce the time between clock edges which in turn reduces the time for metastability resolution, thereby reducing reliability.

The bandwidth of the simple two flip-flop synchroniser is also poor. If the asynchronous subsystem produces data more quickly than the synchronous interface can consume it then the request-in to request-out delay will be two clock cycles. This can be improved upon by using an asynchronous FIFO to buffer the data together with a signal which indicates that there are at least k items to be retrieved [7]. This allows the synchronous side to keep fetching data every clock cycle until there are less than k items left, but this adds complexity to the system.

However, it should be noted that one advantage of this simple scheme is that many asynchronous data streams can be simultaneously synchronised without reducing performance.

3. Stretching the clock

An alternative strategy is to stretch the clock when there is a risk of metastability [1, 8, 10]. Since the clock has to be pausable, we use a calibrated delay-line to generate a clock reference [3]. A crystal

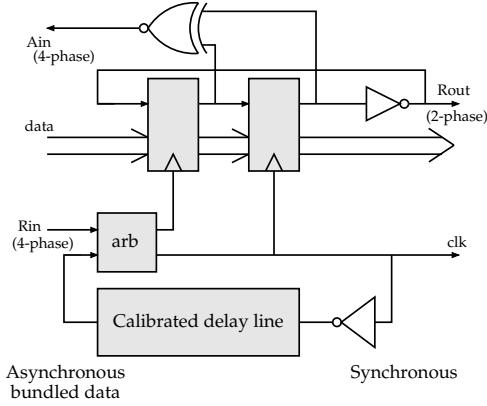


Figure 2. Simple Clock Stretching Circuit

oscillator is used as a timing reference to calibrate the delay-line. The crystal oscillator could not be used directly because it cannot be stopped and restarted quickly.

The basic stoppable clock scheme is depicted in Figure 2. A rising request signal ($Rin+$) from the asynchronous circuit takes a lock out on the clock via a Seitz arbiter [6]. Only when the clock is low will the request signal propagate through the arbiter to clock the first set of data flip-flops. Once this new data is held in the first set of flip-flops and the request falls ($Rin-$) then a positive clock edge may pass through the arbiter which in turn latches data into the second set of data flip-flops. Thus, new data from the asynchronous side will always be held by the first set of data flip-flops before the second set of data flip-flops are clocked.

In the best case, new data arrives shortly before the next clock edge and is held in the first set of flip-flops. At the clock edge, data is transferred to the synchronous environment. This data setup time is typically less than half a clock cycle and is the minimum latency. The maximum latency occurs when new data arrives just after the clock goes high and must wait for the clock to go low before being allowed to latch new data. Thus the maximum latency is one clock cycle which is much better than the two flip-flop synchroniser. Furthermore, data can be transferred every clock cycle without additional buffering. However, unlike the two flip-flop synchroniser, the clock is likely to be delayed by the arbitration process which reduces the performance of the synchronous system.

If there is more than one asynchronous data stream then these streams could take turns to supply data to the synchronous system by using a set of arbitrating call elements [10]. However, this serialises the data transmission which adds further latency.

4. Clock Prediction

Our first improvement to the stretchable clock approach is to predict when the next positive clock edge will occur and to perform arbitration early. In this way the synchronous subsystem has priority over the asynchronous subsystem so the clock should rarely be delayed.

Figure 3 presents an outline of this scheme. The asynchronous/synchronous interface produces data enable signals in response to new data from the asynchronous interfaces. These enable signals are guaranteed to meet setup and hold constraints for the data flip-flops and always occur after the asynchronous data has stabilised so metastability can never arise.

We use a simple clock predictor: when the clock is currently low it will want to go high in the near future. Thus we use the inverted clock to lock out the asynchronous subsystem which usually happens in less than half a clock cycle (see Section 7).

The inverted clock is fed to the timing delay line and a clock pause circuit. These race each other, usually with the clock pause circuit producing $clkallowed$ before the clock edge appears from the delay line. A C-element ensures that both have arrived before a positive clock edge is generated. The clock pause circuit relies on Seitz arbiters [6] which may take an unbounded time to resolve if an internal metastable state arises. However, a more detailed analysis in Section 7 indicates that the clock will only be delayed if metastability resolution takes an unusually long time.

The details of the clock pause circuit are discussed in the next section and the rest of the asynchronous/synchronous interface, which performs handshake decoupling, is presented in Section 6.

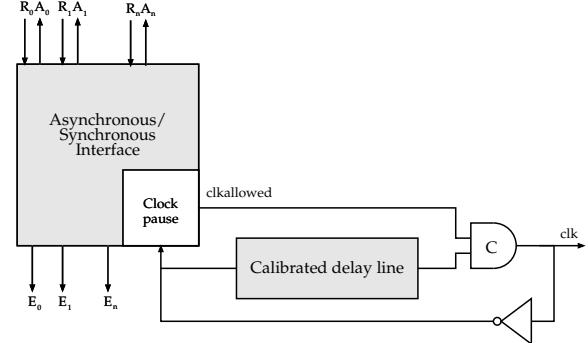


Figure 3. Predictive clock stretching circuit

5. Parallel clock pause circuit

The clock pause block in Figure 4 consists of a number of arbiters, one per asynchronous interface. Each asynchronous interface may make a request to present new data to the clocked system by raising RC_n . The arbiter will only acknowledge this request (raises AC_n) when \overline{clk} is high. If \overline{clk} falls and RC_n rises simultaneously then the arbiter may go metastable internally, but will safely choose one of the requests over the other.

The inverted \overline{clk} predicts that the clock needs to go high soon so it locks out all of the asynchronous interfaces by forcing the arbiters to grant in its favour. This locking process results in $clkallowed$ going high.

When \overline{clk} is high, it releases its hold on the arbiters giving all asynchronous interfaces a window of opportunity to simultaneously supply new data to the synchronous system. The timing of this window can be adjusted by delaying inverted input clock so that the window of opportunity is near (but not too close to) the rising clock edge (clk). However, time must be allowed for the arbitration process to complete early if the clock is not to be delayed.

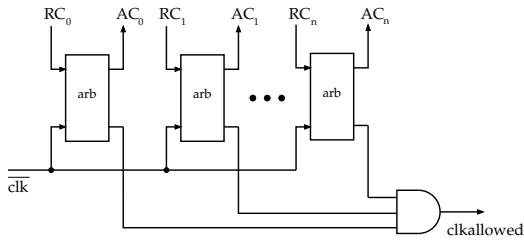


Figure 4. Clock Pause Block

6. Decoupler Circuit

For each asynchronous interface there is a decoupler circuit 5. The decoupler guarantees the performance of the asynchronous/synchronous interface by ensuring that the asynchronous side only holds a lock on its arbiter for the minimum time necessary to ensure correct operation.

The detailed operation is as follows. When new data arrives (New_data_+) a request is made (RC_n+) to the clock pause block which acknowledges with AC_n+ when it is safe to supply the synchronous circuit with new data. When AC_n+ is received it sets the RS flip-flop and enables the data flip-flops so that the new data can be latched on the next clock edge. Shortly after the RS flip-flop is set, the RC_n- transition will release the arbiter in the clock pause circuit. This ensures that RC is high for as short a period as possible

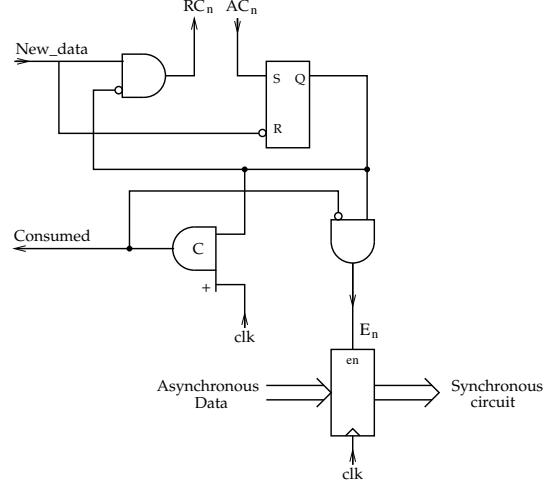


Figure 5. Decoupler circuit

in order to minimise the risk of delaying the system clock.

When the synchronous circuit consumes the data on the rising edge of the clk , the $Consumed$ signal is raised which lowers the enable to the data flip-flops. $Consumed$ going high also allows the asynchronous interface to respond by lowering New_data which results in $Consumed$ going low, all ready to receive some more data.

7. Analysis of the interface

To ensure that it is unlikely that the New_data signal will delay the clock we need to test the critical path from New_data followed by clk going low to the $clockallowed$ signal going high, i.e. how long the clock pause circuit can delay the $clockallowed$ signal. SPICE simulation of a $0.35\mu m$ CMOS implementation indicates that this critical path is 0.98ns provided $clk-$ is sufficiently after New_data+ that the arbiter does not go metastable. If metastability does occur then the resolution time depends on the exact timing of the input signals, their rise time and the gain inside the arbiter.

If the tuned delay line were set at 2.5ns (to generate a 200MHz clock) then the asynchronous interfaces would be allowed to supply data when the clk is high (for 2.5ns). When clk goes low, arbitration begins, which we know will take 0.98ns. Even if metastability resolution takes up to 1.5ns, the clock (clk) will not be delayed. Thus, in the best case it will take around 0.5 clock cycles to transfer data from the asynchronous interface to the synchronous system, and in the worst case 1.5 clock cycles. If the asynchronous interface is eager (e.g. it is supplying data buffered in a FIFO) then it is able to supply new

data every clock cycle.

The decoupler circuit is an asynchronous finite state machine. Various internal delay assumptions are made but the external signals are delay insensitive. Analysis of the internal delays (using our in house tool) reveals that all assumptions satisfy the rule “any two gate delays takes longer than any one gate delay”. The most critical delay is the data flip-flop enable signal which must be set before the clock can go high. If necessary an additional delay margin may be added to the feedback from the RS flip-flop to delay RC-.

8. Synchronous producer to an asynchronous consumer

So far this paper has talked about an asynchronous producer and a synchronous consumer. However, the converse is quite straight forward since the synchronous subsystem can output data at any time provided the asynchronous side is *ready*. This *ready* state is simply an asynchronous input to the synchronous system and this just requires an instance of our asynchronous/synchronous interface.

9. Conclusion

This paper has presented a low latency, high bandwidth and reliable interface between asynchronous and synchronous subsystems. Clock stretching is used to prevent metastability when the synchronous system samples data from the asynchronous environment. However, unlike other designs in the literature, our interface is capable of granting many asynchronous data requests in parallel. Furthermore, arbitration between the asynchronous and the synchronous sides is undertaken in advance of the next positive clock edge. This ensures that the clock is almost never delayed so the synchronous system runs at its full rate.

Communication latency is between 0.5 and 1.5 clock cycles which compares favourably with the two flip-flop synchroniser where latency is between 1 to 2 clock cycles. If the asynchronous subsystem is eager then our design will ensure that data can be transferred on every clock cycle. This is only possible with the two flip-flop synchroniser if additional buffering is used.

From a reliability point of view, our design will never fail due to a metastable condition, whereas the two flip-flop synchroniser can fail if insufficient time is allowed for the metastable condition to be resolved.

References

- [1] David S. Bormann and Peter Y. K. Cheung. Asynchronous wrapper for heterogeneous systems. In

Proc. International Conf. Computer Design (ICCD), October 1997.

- [2] William J. Dally and John W. Poultom. *Digital Systems Engineering*. Cambridge University Press, 1998.
- [3] George Taylor, Simon Moore, Steev Wilcox and Peter Robinson. An on-chip dynamically recalibrated delay line for embedded self-timed systems. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, April 2000.
- [4] Howard W. Johnson and Martin Graham. *High-Speed Digital Design — A Handbook of Black Magic*. Prentice Hall, 1993.
- [5] Fred U. Rosenberger, Charles E. Molnar, Thomas J. Chaney, and Ting-Pien Fang. Q-modules: Internally clocked delay-insensitive modules. *IEEE Transactions on Computers*, C-37(9):1005–1018, September 1988.
- [6] Charles L. Seitz. System timing. In Carver A. Mead and Lynn A. Conway, editors, *Introduction to VLSI Systems*, chapter 7. Addison-Wesley, 1980.
- [7] Jakov N. Seizovic. Pipeline synchronization. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 87–96, November 1994.
- [8] Allen E. Sjogren and Chris J. Myers. Interfacing synchronous and asynchronous modules within a high-speed pipeline. In *Advanced Research in VLSI*, pages 47–61, September 1997.
- [9] Ivan E. Sutherland. Micropipelines. *Communications of the ACM*, 32(6):720–738, June 1989.
- [10] K.Y. Yun and A. E. Dooley. Pausible clocking based heterogeneous systems. *IEEE Transactions on VLSI Systems*, 7(4):482–487, December 1999.

Acknowledgements

The authors would like to acknowledge the support of EPSRC grant GR/L86326, Cambridge Consultants Ltd and AT&T Labs Cambridge.