

Channel Communication Between Independent Clock Domains

Simon Moore, George Taylor, Bob Mullins, Peter Robinson
Computer Laboratory, University of Cambridge, UK
Simon.Moore@cl.cam.ac.uk

Abstract

Reliable low-latency channel communication between independent clock domains may be achieved using a combination of clock pausing techniques, self-calibrating delay lines and an asynchronous interconnect. Such a scheme could be used for point-to-point communication in a globally asynchronous locally synchronous (GALS) system. These ideas could then be extended to incorporate an asynchronous bus or other chip-wide network architectures.

Clock pausing is used to guarantee that metastability will never result in failure. Arbitration between channel communication and the local clock is performed concurrently so that metastability resolution will rarely delay the clock.

1 Introduction

Globally Asynchronous, Locally Synchronous (GALS) systems offer a high performance and low power solution for system on a chip implementors [3]. Clocked building blocks can be integrated onto one chip with independent clocks for each block and an asynchronous interconnect between them.

Synchronising to asynchronous data is a well known problem which can be crudely resolved by latching the data at least twice to allow time for metastability in the latches to resolve. 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.

An alternative strategy is to stretch the clock when there is a risk of metastability [1, 7, 8]. These schemes rely on generating the clock from a delay line because such a clock is simple to stop by gating the clock pulse. Self calibrating delay lines may be used to provide an accurate timing reference [2].

In this paper we describe a reliable, low latency and high bandwidth channel communication mechanism which may be positioned between independently clocked synchronous domains.

The channel itself relies on asynchronous logic techniques.

To assist explanation, the design is partitioned into two parts. First we will describe our mechanism for reliable data transfer between an asynchronous producer (one end of the asynchronous channel) and a synchronous consumer. Then we will discuss the synchronous producer which sends data to the asynchronous channel. These two parts are then combined and we look at additional decoupling and other techniques required to maximise bandwidth. Detailed simulation results are then presented and conclusions drawn.

2 Asynchronous producer, synchronous consumer

Figure 1 presents an interface between an asynchronous producer and a synchronous consumer which is based on our earlier work [5, 6]. When no data is being presented, the output clock (clk) is inverted and then fed to a calibrated delay line [2] and to one input of an arbiter. The arbiter then grants in the favour of the clock circuit and is merged with the output from the delay line. This oscillatory process continues and a stable clock signal is produced.

Data from the asynchronous producer is signalled by a transition in the req signal (see Figure 2). This will result in the arbiter granting in the favour of the asynchronous interface when the clock (clk) is high. If the clock rises ($\text{clk}+$) and req changes around the same time then the arbiter may go metastable. This is safe since the guarantees that the clock cannot continue until it is granted nor will the input req signal be propagated.

Once the arbiter grants in favour of the input request (req), data is latched which in turn releases the hold on the arbiter. Now that the data has been safely latched, it is safe for the synchronous consumer to be presented with a rising clock edge which latches the incoming data in the final set of latches. The incoming req signal is also latched in the final set of latches which is also fed-back as an acknowledge signal.

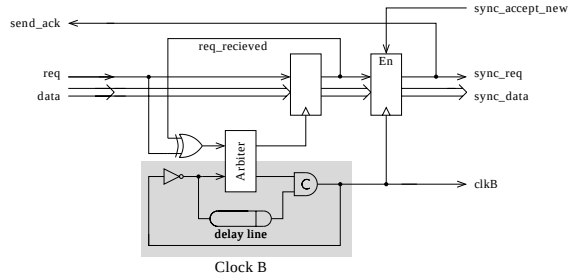


Figure 1: Interface between an asynchronous producer and a synchronous consumer

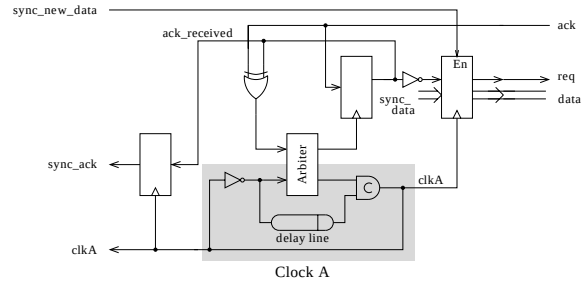


Figure 3: Interface between a synchronous producer and an asynchronous consumer

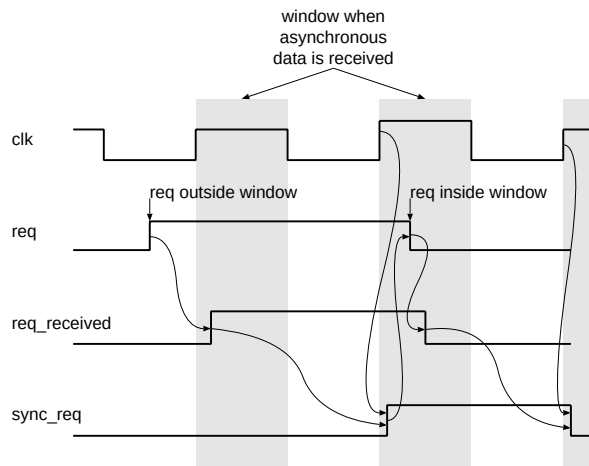


Figure 2: Request in to request out traces

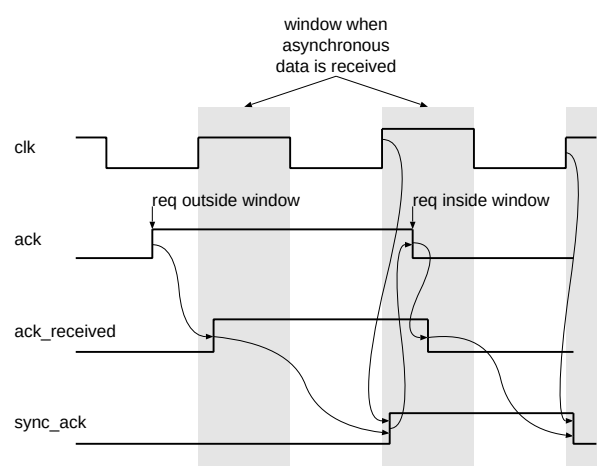


Figure 4: Request in to request out traces

3 Synchronous producer, asynchronous consumer

Transmitting data from the synchronous domain to the asynchronous domain requires that the synchronous system knows when the asynchronous domain is able to accept data. Thus, the primary difficulty is safe transfer of this control signal from the asynchronous domain to the synchronous one. Therefore, the circuit we require (see Figure 3) is similar to the asynchronous producer, synchronous consumer circuit described in the previous section. The relationship between the incoming asynchronous acknowledge (ack) and the outgoing synchronous acknowledge (sync_ack) in Figure 4, is identical to the request (req) and synchronised request (sync_req) presented in Figure 2.

The only difference is the transmission of data from the synchronous world to the asynchronous world. A 2-phase request signal is bundled with the data to indicate to the asynchronous

system that new data is available. Provided the synchronous state machine always waits for an sync_ack between sending data items, the req will always toggle and data will be sent correctly.

4 Channel communication with no buffering

The circuits from the previous two sections may be glued together to form a simple channel communications structure. However, as can be seen from Figure 4, the synchronous producer will have to wait at least one clock cycle between sending data and receiving a corresponding sync_ack signal. This will limit the bandwidth to a maximum of one data item every second clock cycle. The average bandwidth is typically worse than this because the outgoing req has to pass right through the consumer control logic, back out via send_ack and back through control logic in the producer before sync_ack is seen.

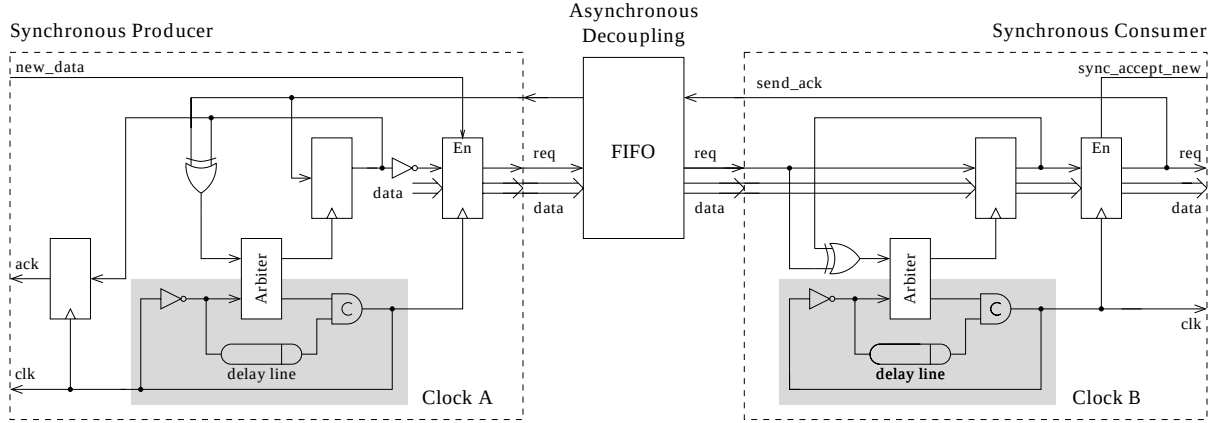


Figure 5: Channel communication with FIFO buffering

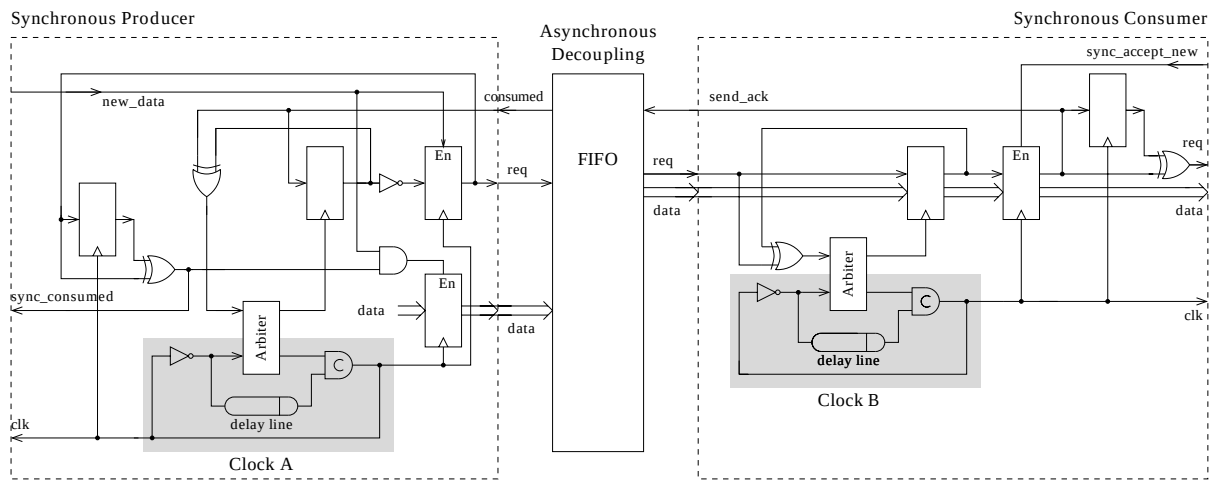


Figure 6: High bandwidth channel communication scheme

5 Channel communication with buffering

Adding an asynchronous FIFO between the producer and consumer (see Figure 5) decouples the $req \rightarrow sync_ack$ path. This allows the producer to reliably send just one data item every second clock cycle.

For the producer to be able to transmit a data item every clock cycle we have to make the following assumption: if the first element of the FIFO is empty then any data emitted by the synchronous producer will be safely stored before the next clock cycle. This is a very safe timing assumption. From this assumption it is clear that in place of the ack signal from the FIFO we require a *first element of FIFO empty* signal which we will call *consumed*.

This new scheme is presented in Figure 6.

The synchronous producer may generate new data every time there is an outgoing transition on req . If the FIFO becomes full then no ack will be received from the FIFO which in turn means that no req will be produced. Under this condition, data must be held in the output latch by deasserting the enable signal on this latch.

6 Simulation results

Simulations were performed using Verilog with SDF annotation using the the VST/UMC 0.25 μ m library timing and gate loading model. The consumer runs at a fixed frequency of 50MHz, the producer at a varying frequency up to 200MHz.

We tested the simple control scheme (Figure 5) for various FIFO depths. When the FIFO is of size zero, it corresponds to the circuit in Figure 5 with the FIFO removed. The detailed simulations were

undertaken and the bandwidth measurements are shown in Figure 7. Making the FIFO deeper than a depth of two does not improve performance further.

The more complex control scheme (Figure 6) requires a FIFO depth of at least two items. However, making the FIFO deeper does not improve performance further. The bandwidth results are presented in Figure 8.

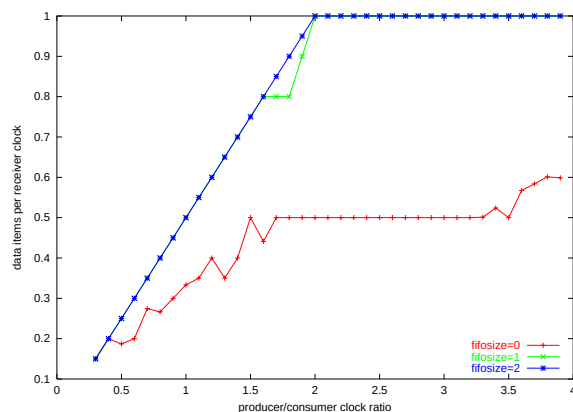


Figure 7: Throughput of simple channel as FIFO size varies

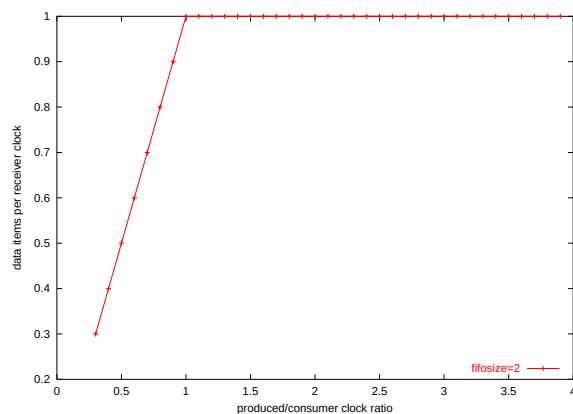


Figure 8: Throughput of fast channel for the minimum sized (two item) FIFO

7 Conclusions

We have presented a asynchronous channel communication mechanism which allows low latency, high bandwidth and reliable data transfer between independently clocked synchronous domains. This mechanism could form the basis of a globally asynchronous, locally synchronous

(GALS) chip-level architecture in order to meet the challenges of deep submicron CMOS design.

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] 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.
- [3] A. Hemani, T. Meincke, S. Kumar, A. Postula, T. Olsson, P. Nilsson, J. Öberg, P. Ellervee, and D. Lundqvist. Lowering power consumption in clock by using globally asynchronous, locally synchronous design style. In *Proc. ACM/IEEE Design Automation Conference*, 1999.
- [4] Howard W. Johnson and Martin Graham. *High-Speed Digital Design — A Handbook of Black Magic*. Prentice Hall, 1993.
- [5] S. W. Moore, G. S. Taylor, P. A. Cunningham, R. D. Mullins, and P. Robinson. Self-calibrating clocks for globally asynchronous locally synchronous systems. In *Proc. International Conf. Computer Design (ICCD)*, September 2000.
- [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 subsystems. In *Asynchronous INTERfaces: tools, techniques, and implementations (AINT)*, July 2000.
- [7] 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.
- [8] K.Y. Yun and A. E. Dooply. 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.