Point to Point GALS Interconnect

Simon Moore, George Taylor, Robert 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 can be used for point-to-point communication in a globally asynchronous locally synchronous (GALS) system, a possible methodology for managing the predicted increase in clock domains.

We present interface wrapper circuits which permit communication between a locally synchronous producer and a locally synchronous consumer via an asynchronous interconnect. Such interfaces can also be used to mix asynchronous and synchronous modules. 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.

Simulation results show that the maximum performance of one data item per consumer clock cycle is achieved when the producer:consumer clock ratio is equal or greater to one. This communication mechanism is suited to other asynchronous interconnect methods which offer low power and high performance.

1. Introduction

The Semiconductor Industry Association (SIA) roadmap and Electronic Design Automation Industry Council predict that the number of clock domains on a silicon chip will grow rapidly. This is due to wire delays becoming harder to predict due to process variation, deep-sub micron effects and dynamic effects such as cross-talk.

An alternative to the challenge of clocking many time zones with controlled skew is a change to more globally asynchronous architecture. From a theoretical standpoint, circuits with delay-insensitive asynchronous interfaces can be composed to produce large systems which are guaranteed to be functionally correct. From a design perspective this composition property eases the worry of timing closure for large systems. Thus, the SIA Roadmap recognises that, by 2007, asynchronous techniques will be used in many designs. However, meeting a performance target may still require multiple design iterations.

We are also entering the systems-on-chip (SoC) era where circuit building blocks from a number of design houses (intellectual property blocks or IP-blocks) are purchased by a systems builder and integrated onto a single chip. In some ways this is similar to building systems by purchasing off the shelf ICs and integrating them via a PCB. However, SoC results in lower cost mass market products with much lower power requirements. Communication between building blocks of a SoC is a complex problem particularly when a range of clocking strategies have to be tailored to each building block in order to obtain the required performance within a power budget.

Clocked circuit design is a mature method with good tool support. Clocked circuits with global synchronisation are becoming impractical for large, deep submicron CMOS designs, where for smaller circuits assumptions about clock distribution are still valid. Given the investment in clocked tools and techniques, clocked design in the small will continue to be an attractive technique in industry. One methodology which builds on this investment is that of globally asynchronous, locally synchronous (GALS) systems where asynchronous communication techniques are used for long distance communication between individually clocked subsystems.

2. Globally Asynchronous, Locally Synchronous Systems

Globally Asynchronous, Locally Synchronous (GALS) systems may offer a solution for system on a chip implementors seeking good performance and low power consumption [6]. 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 [7]. 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 [11, 13, 3, 2, 14, 16, 10]. Such a scheme relies 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 [5].

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 divided into several parts. Section 3 describes our mechanism for reliable data transfer between an asynchronous producer (the output of the asynchronous channel) and a synchronous consumer. Section 4 discusses the synchronous producer which sends data to the asynchronous channel. Section 5 combines these two parts and Section 6 looks at additional decoupling and other techniques required to maximise bandwidth. Section 7 presents a variation in which the producer/consumer can enter 'sleep mode' with the clock stopped. Simulation results are then presented and conclusions drawn.

3. Asynchronous producer, synchronous consumer

Figure 1 presents an interface between an asynchronous producer and a synchronous consumer which is based on our earlier work [8, 9]. When no data is being presented, the output clock (clkB) is inverted and then fed to a calibrated delay line [5] 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. Examples of how delay lines may be constructed from standard cells are provided in the Annex.

Data from the asynchronous producer is signalled by a transition in the req signal (see Figure 2); the point-to-point channel uses a two-phase signalling scheme. This will result in the arbiter granting in the favour of the asynchronous interface when the clock (clkB) is high. If the clock falls and req changes around the same time then the arbiter may go metastable. This is safe since the arbiter guarantees that the clock cannot continue and the input req signal can not be propagated until the metastability has resolved.

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 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.
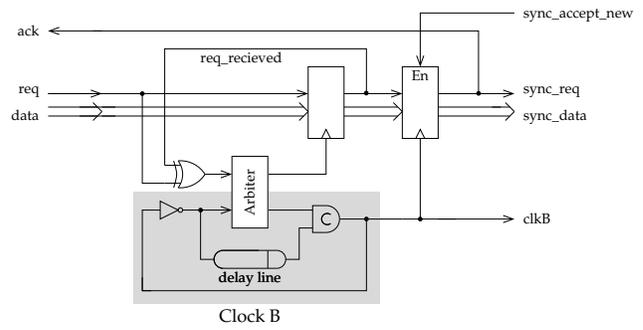

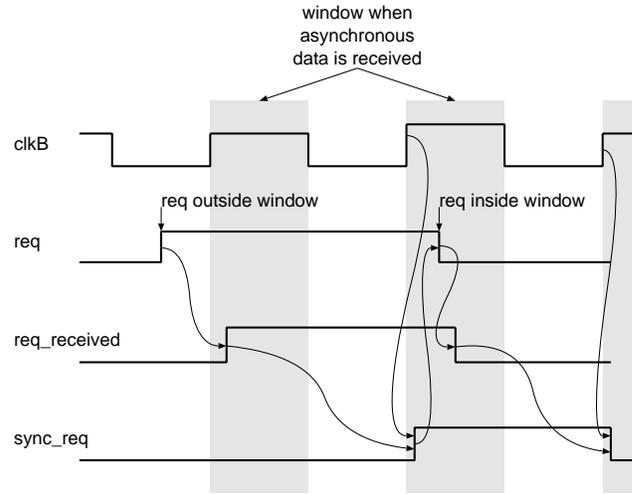**Figure 1. Interface between an asynchronous producer and a synchronous consumer**

Figure 2. Request in to request out traces

4. 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.

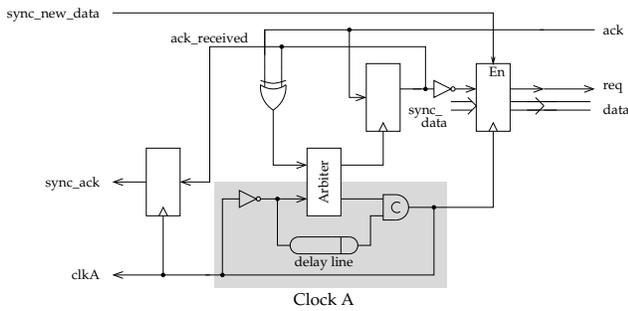The only difference is the transmission of data from the synchronous world to the asynchronous world. A

Figure 3. Interface between an synchronous producer and an asynchronous consumer
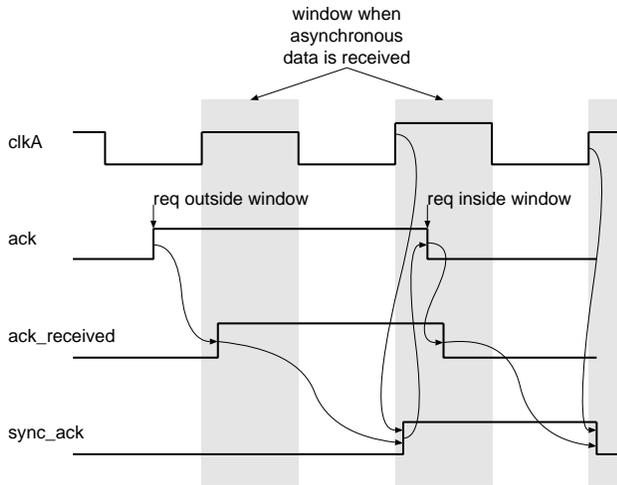

Figure 4. Acknowledge in to Acknowledge out traces

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.

5. Channel communication with no buffering

The circuits from the previous two sections may be concatenated together to form a simple asynchronous communication channel between two clocked domains. 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 ack and back through control logic in the producer before sync_ack is seen.

6. Channel communication with buffering

Adding an asynchronous FIFO between the producer and consumer (see Figure 5) decouples the req→sync_ack path. This allows the producer to reliably send 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 *input element of FIFO is empty* signal which we will call consumed.

This new scheme is presented in Figure 6, with two-phase to level sensitive conversion of sync_req and sync_consumed now explicitly shown. The synchronous producer may generate new data every time there is an outgoing transition on req. If the FIFO becomes full then no consumed 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.

7. Variation with sleep mode

For some applications it may be desirable for the synchronous consumer to enter a sleep state with the clock stopped until new data input arrives.

Figure 7 shows an extension of the circuit from Figure 1. A second arbiter is inserted to permit the clock to be stopped; note that both arbitrations are concurrent and that the upper grant output of the additional arbiter is not required. When the synchronous consumer wishes to sleep it raises sync_sleep causing the clock to stop. The XOR clears the now set DFF once new data has been latched by the interface. The DFF could be replaced with an SR latch where reset has priority. This circuit extension is also applicable to the high-bandwidth variation from Figure 6.

A similar arrangement applies for the producer, shown in Figure 8. Here the synchronous producer can request to sleep until the previously sent data has been read.

8. Multiple interfaces

A GALS block with more than one input/output interface can be constructed by adding additional arbiters in parallel. The single arbiter is replaced with multiple aribters, see Figure 9 and an AND gate, much like the scheme used in
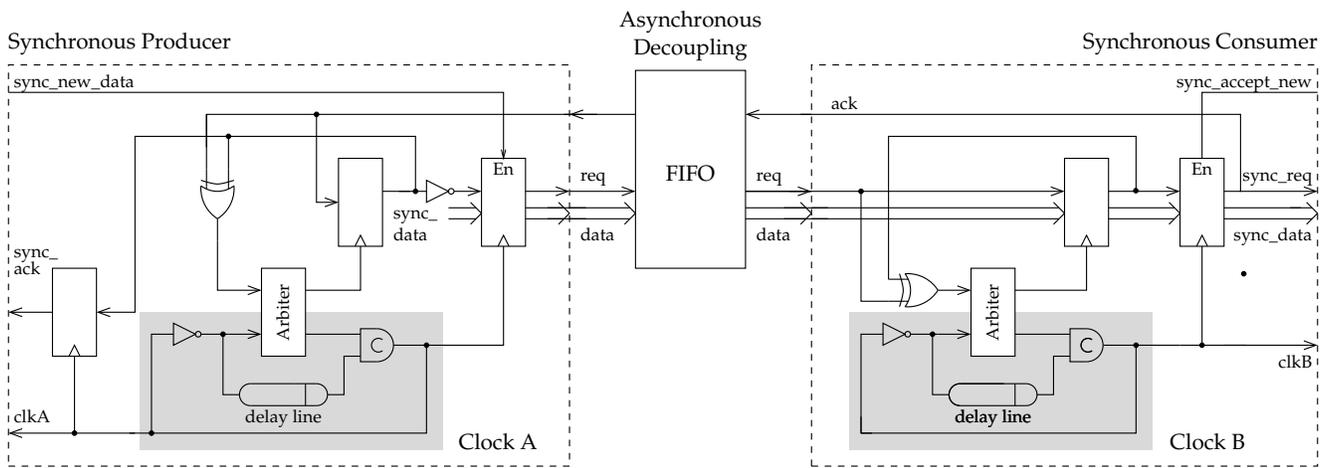
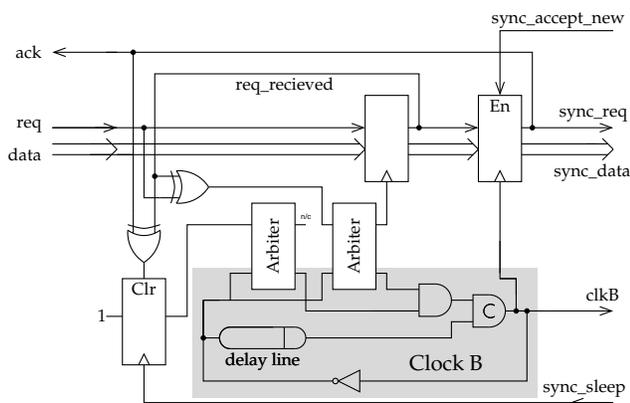**Figure 5. Channel communication with FIFO buffering**

Figure 7. Synchronous consumer interface with sleep feature
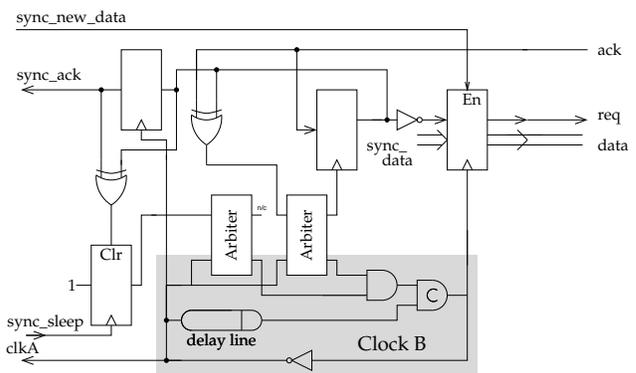

Figure 8. Synchronous producer interface with sleep feature

Section 7. Each arbiter arbitrates between a GALS interface and the single clock generator. Such a scheme permits multiple data items to be input or output in each clock cycle, albeit with an increased probability of the clock being paused.

9. Results

Simulations were performed using Verilog with SDF annotation using the VST/UMC 0.25μm timing library with wireload model. The consumer runs at a fixed frequency of 50MHz, the producer at a varying frequency up to 200MHz, obviously higher frequencies are possible depending on the complexity of the producer and consumer.

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 in Figure 10. Making the FIFO more than two slots long 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 11.

The calibrated delay line and arbiter to stop and start the clock has been fabricated on a 0.35μm process. In this test a 25-80MHz clock is produced using a 32kHz watch crystal as a calibration reference, calibration takes place once per second. This clock has been successfully used to clock both a 16 bit microcontroller and a UART serial interface. A revised version with finer adjustment steps has
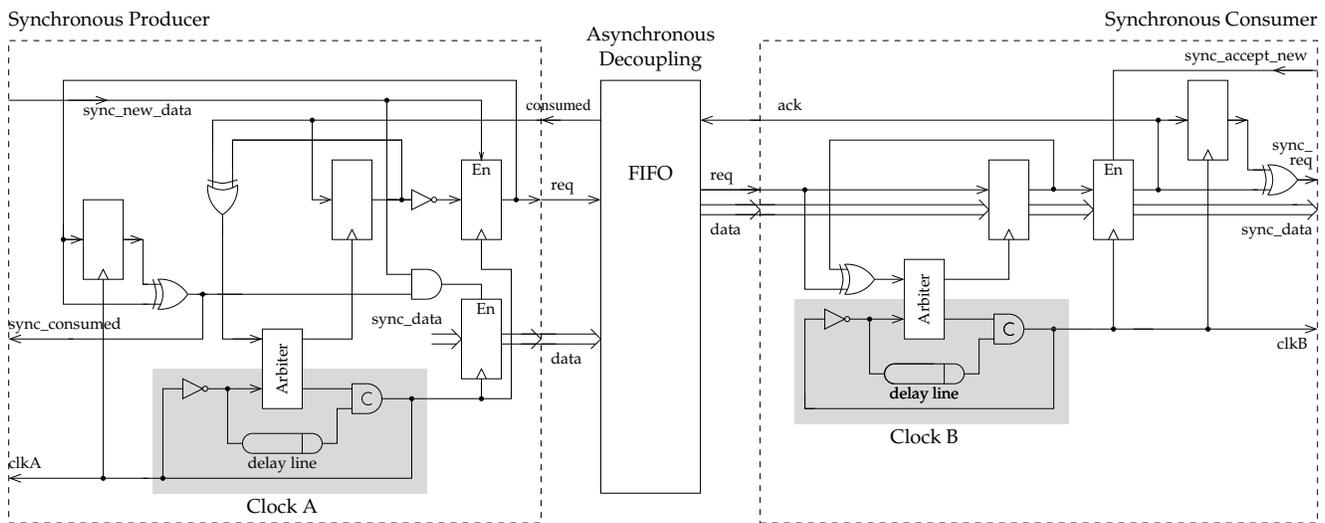
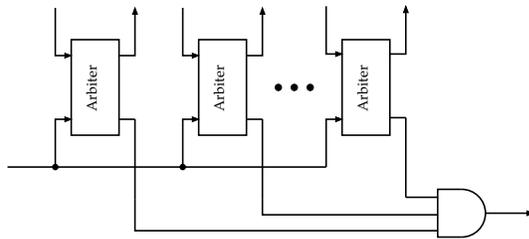**Figure 6. High bandwidth channel communication scheme**

Figure 9. Concurrent arbiters

been submitted to a $0.18\mu m$ process.

10. Review

It is worth noting that some special cases of the GALS interface exist where arbitration is not required.

Case 1—Asynchronous consumer always ready and synchronous producer: Here the synchronous producer simply sends a request and data to the consumer. No acknowledge from the consumer is used. Arbitration and clock pausing are not required. Typically a consumer would always be ready if there was only one input port and it could consume the data within the producer clock cycle. An example of this is a memory device.

Case 2—Asynchronous producer always able to produce data and synchronous consumer: This is the converse of case 1. Arbitration and clock pausing is not required, the consumer clocks data out of the asynchronous FIFO.

Case 3—Synchronous producer and slave consumer. Here the synchronous producer clock pauses until the consumer has acknowledged receipt of the data and
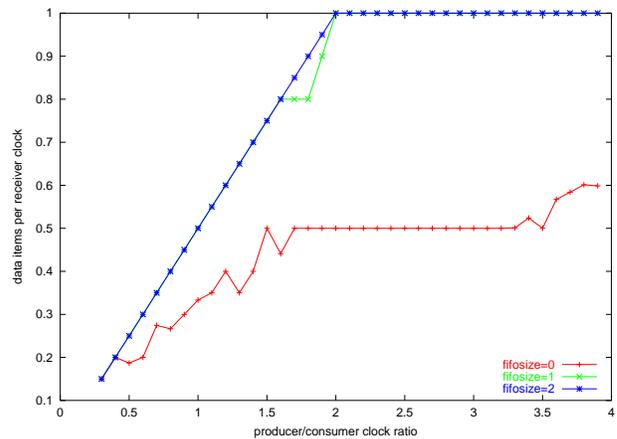

Figure 10. Throughput of simple channel as FIFO size varies

returned any response, for example a microcontroller accessing a slow delay-matched memory. Clock pausing is required but not arbitration. A similar principle can be used to embed asynchronous modules inside an otherwise synchronous pipeline [14]. Additionally [14] discusses the impact of the clock buffer insertion delay which we have not addressed here.

So far GALS interfaces with zero probability of failure have been discussed, however, this is at the expense of clock pausing, possibly with arbitration. Additionally failure to meet a timing deadline due to metastability resolution may be considered a failure in some real-time systems.
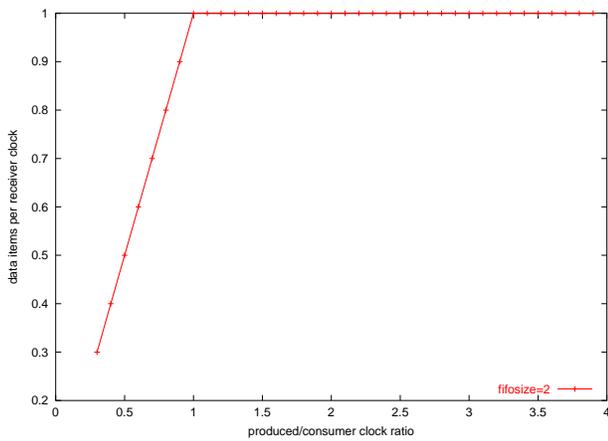
Figure 11. Throughput of fast channel for the minimum sized (two item) FIFO

An alternative is to use two cascaded registers for synchronisation but instead of synchronising the data synchronising some control signals indicating the state of the FIFO between producer and consumer [4]. This permits truly free running clocks and the probability of synchronisation failure is linearly reduced by reducing the number of synchronisations without reducing bandwidth. During continuous steady-state data transmission the probability of synchronisation failure is zero, however, such a state may be hard to maintain. This scheme involves extra latency and a startup cost when the FIFO becomes empty.

Each GALS block has a bundled data interface and the delay line for the local clock delay matches the worst case combinational logic path within the synchronous block. This could be viewed as a coarser scale version of standard bundled data design, where a unique delay is used to match the worst case path in each of many small combinational logic blocks. Design styles with even more localised delay assumptions include IPCMOS [12] which uses a pulse generator to strobe latches, and GASP with self-reset [15].

The GALS methodology is well suited to other asynchronous interconnect methods. For simplicity, in this paper bundled data has been used, the request signal is bundled with the data and is assumed to not arrive earlier than the data. Such a scheme is unable to exploit the widely varying data dependent delays for data transmission. At the extreme case, compare one bit going high whilst the others go low with all bits changing in the same direction.

Alternative interconnect could use dual-rail or one-of-four encoding. Such interconnect is truly self-timed, the consumer is able to proceed as soon as data arrives. For large interconnect capacitances the one-of-four encoding scheme offers half the transition power and a lower latency that the bundled and dual-rail schemes [1]. The GALS interface structure would remain the same with the consumers request input supplied from the interconnect completion detection logic. Further bandwidth improvement might be achieved by adding FIFO stages and distributing these between the producer and consumer. Finally it should also be possible to apply the GALS principle to a bus rather than point-to-point link.

11. Conclusion

We have presented an 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 sub-micron CMOS design.

Annex: Delay line circuits

To achieve a very fine adjustment in the delay our delay lines are composed of two sections, fine and coarse adjustment sections. This is a variation of a previous scheme [8] where fine and coarse adjustment were integrated. The idea is that the fine delay section provides a total delay range of about one delay step in the coarse delay section. This scheme is shown in Figure 12.

The coarse delay line is based upon blocks of inverter delays and a binary scheme, see Figure 13. The fine delay adjustment step requires to be smaller than a normal standard cell gate delay, two possible methods are given here. The first, shown in Figure 14, exploits changing the pull-up resistance in a simple AO gate shown in Figure 15. The second involves having a number of tristate buffers in parallel, the more of which are enabled, the higher the drive strength and thus shorter the delay.

Adjusting the coarse delay should only be performed whilst the clock is stopped and the delay line contains logical zeros, otherwise an extra pulse might be inserted in the feedback ring. Various versions of these delay lines have been included on a 0.18μm test chip submitted for fabrication.

References

[1] W. J. Bainbridge and S. B. Furber. Delay insensitive system-on-chip interconnect using 1-of-4 data encoding. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, March 2001.

[2] David S. Bormann and Peter Y. K. Cheung. Asynchronous wrapper for heterogeneous systems. In *Proc. International Conf. Computer Design (ICCD)*, October 1997.
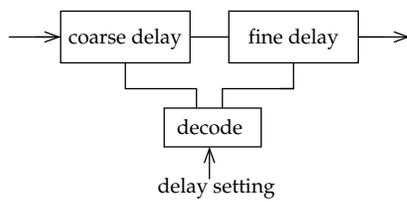
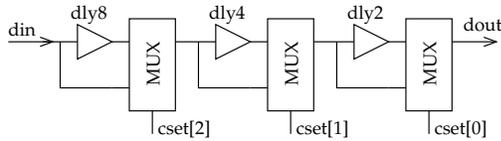**Figure 12. Block view of delay line**
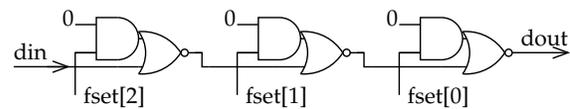

Figure 13. Coarse section

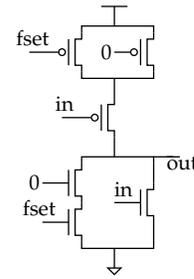**Figure 14. Fine adjustment: AO gate version**
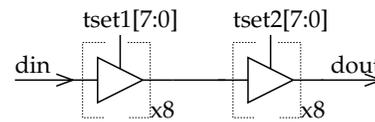

Figure 15. AO gate

Figure 16. Fine adjustment: Tristate version

[3] Daniel M. Chapiro. *Globally-Asynchronous Locally-Synchronous Systems*. PhD thesis, Stanford University, October 1984.

[4] T. Chelcea and S. M. Nowick. A low-latency FIFO for mixed-clock systems. In *Proceedings of the IEE Computer Society Annual Workshop on VLSI (WVLSI'00)*, April 2000.

[5] 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.

[6] 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.

[7] Howard W. Johnson and Martin Graham. *High-Speed Digital Design — A Handbook of Black Magic*. Prentice Hall, 1993.

[8] 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.

[9] 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.

[10] Jens Muttersbach, Thomas Villiger, and Wolfgang Fichtner. Practical design of globally-asynchronous locally-synchronous systems. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 52–59, April 2000.

[11] Miroslav Pečhouček. Anomalous response times of input synchronizers. *IEEE Transactions on Computers*, 25(2):133–139, February 1976.

[12] S. E. Schuster. Asynchronous interlocked pipelined cmos (IPCMOS) circuits operating at 3.3-4.5GHz. In *Asynchronous INTerfaces: tools, techniques, and implementations (AINT)*, July 2000.

[13] Charles L. Seitz. System timing. In Carver A. Mead and Lynn A. Conway, editors, *Introduction to VLSI Systems*, chapter 7. Addison-Wesley, 1980.

[14] 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.

[15] I. Sutherland and S. Fairbanks. GasP: a minimal FIFO control. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, March 2001.

[16] K.Y. Yun and A. E. Dooply. Pausible clocking based heterogeneous systems. *IEEE Transactions on VLSI Systems*, 7(4):482–487, December 1999.

Acknowledgements