A FIFO buffer with non-blocking interface*

A. Yakovlev[†] D.J. Kinniment, F. Xia, A.M. Koelmans
Department of Computing Science
Department of Electronic Engineering
University of Newcastle upon Tyne
NE1 7RU England

Abstract. The design of a FIFO buffer supporting fully asynchronous, non-blocking, read and write operations is presented. The circuit consists of a standard self-timed FIFO surrounded by two special interface blocks, In and Out, which together ensure temporal independence between the reading and writing sides. These blocks are synthesised from their Signal Transition Graph (STG) specifications using an asynchronous circuit synthesis tool, `petrify`. The proposed solution is compared and contrasted against "reference data" mechanisms. It is shown to be more appropriate for "profile data" applications, such as image or speech data transmission.

1 Introduction

Real time systems often need to provide data communication between subsystems with independent timing of their send and receive procedures. Similar problems arise in modern VLSI chips where new forms of asynchronous communication between modules need to be found to sustain various technological advances, such as, e.g., ability to accommodate subsystems with different clocks on the same chip.

For such applications, any scheme that involves temporal blocking between a pair of Writer and Reader processes may be unacceptable. Here it is assumed that finite shared memory is used between Reader and Writer. For finite systems of this type, two necessary consequences of providing temporal independence for Reader and Writer are *data loss* and *re-reading of old data*. In other words, Writer must be allowed to skip or overwrite data which has not been read by Reader, and that Reader must be allowed to re-read old data it has already obtained in the previous cycle of operation.

In many applications such skipping and re-reading are not critical failures but tolerable disadvantages that should be minimised. The critical property is *data coherence*, which requires that Reader always obtains complete items of data, that is both Writer and Reader never access the same data slot simultaneously.

Other often-cited properties are *data freshness* and *data sequencing*. The former specifies that Reader always obtains new data when it is available. It matters for the so-called *reference data* applications, where, for example, data may carry the current coordinates of a moving object or the current clock value. In those circumstances the relative value of data is reduced with time as it stays longer in transit between Reader and Writer. An example of this type of asynchronous communication is Simpson's four-slot mechanism [1]. This mechanism does not however provide the design flexibility to influence the data loss, re-reading and sequencing properties of implementations.

Our goal here is to *minimize the loss of data* in the channel and to preserve *data sequence*. An error in sequencing is the reading of an older item after a newer item has been already read. This sort of data is called here "profile data", and it is clearly more context-sensitive than "reference data". Image or speech data processing would be typical applications. Due to the presence of a FIFO buffer in the design, it provides some means of tuning the data loss and re-reading rates while providing data coherence and sequencing. Data freshness requirements have been relaxed to allow a measure of latency (which is known to be dependent on a particular type of FIFO architecture [8]) into the mechanism.

*Work is supported by EPSRC (project COMFORT, GR/L93775).
†email contact: alex.yakovlev@ncl.ac.uk

2 A FIFO buffer with non-blocking interface

The classical bounded self-timed FIFO buffer [2] helps to smooth out any temporary differences in the rate between data writing and reading but is blocking. If the average speeds of Writer and Reader differ the buffer will start blocking the activity of the faster party.

We have designed a FIFO buffer with non-blocking Writer and Reader interfaces. In this design, if Writer attempts to write when the buffer is full, the data will be 'skipped'. The skipped data, if tagged appropriately, may in principle be collected for future use but such functionality is outside the scope of this circuit. Similarly, if the buffer is empty while Reader attempts to read, the buffer will send a special acknowledgement to Reader indicating that the latter can use the previously read data item. The structure of the buffer is shown in Figure 1. It has a fairly standard handshake interface with Writer and Reader via the In and Out blocks. The use of handshakes in In and Out is purely for convenience and does not imply any blocking synchronisation, as the following conditions for the In interface show (similar conditions apply for the Out interface).

Real-Time (RT) conditions for In:

The delay between the WrReq event and WrAck event must be bounded. This bound is determined only by the "local" parameters of the circuit, such as the critical cycle of the operation of the In interface and perhaps the propagation delay of the first stage of the self-timed pipeline when that stage is empty.

The operation of the In and Out blocks can be defined using Signal Transition Graphs (STGs) [8]. STGs are a specific interpretation of Petri nets [4], in which events are associated with rising and falling signal transitions. In this way STGs are like timing diagrams. The STG model for the In interface is shown in Figure 2. This model contains special semaphore (mutual exclusion) actions to protect the critical sections which change/test the state of the full/empty flag in the circuit. This semaphore is implemented through a standard 2-input 2-output mutex element (also known as 2-way arbiter) [6]. The circuit synthesised with the `petrify` tool [3] is shown in Figure 3. The design of the Out block is described in [5]. An interconnection between the control logic and the data path in the new buffer is shown in Figure 4. It is assumed here that the data path latches are edge-triggered.

The worst case signal transition path for Writer (similar analysis for Reader is in [5]) corresponds to the situation when the WrReg+ transition occurs concurrently with the arrival of InAck+ from the first stage of the pipe (after the previous data has been sent with InReq+); the InAck+ must then win the mutex. The critical sequence of events (cf. Figure 2) will be as follows: WrReq+, InAck+, r1+, r2+, g2+, Inreq-, InAck-, r2-, g2-, g1+, InReq+, WrAck+, WrReq-, r1-, g1-, dummy, WrAck-. The actual delay of this path can be estimated from the parameters of the circuit implementation. It is clear that although the circuit involves an arbitration element, it is not possible for the "pipe side" to request (and hence win) arbitration more than once without having executed at least one cycle of the (WrReq, WrAck) handshake. Thus, Writer's process can only be "delayed" at most for one pipeline handshake half-cycle, between Inreq- and InAck-. This period occurs in a predictable time because, as can be seen from the implementation of a self-time pipeline stage (Figure 4), the resetting phase of the stage does not depend on the state of the subsequent stage (it is a sufficiently decoupled control); it is virtually the delay of a C-element. This is independent of Reader.

The only "unknown" factor in the above path is the delay caused by potential metastability due to arbitration between the requests r1+ and r2+. In practice, the bound of this delay can be estimated conservatively with extremely high probability [7]. In any case, metastability resolution time for a bistable must be taken into account in any solution with real-time conditions. The above analysis shows that the design satisfies the real-time condition outlined in the beginning of this section. The "write skip" path can be easily traced in the STG and it also meets the real-time conditions.

Analysis in [5] shows that this design provides a measure by which the data loss and re-reading rates can be "tuned". By varying the size of the FIFO it is possible to control these rates to some extent, depending on the statistical properties of the Writer and Reader cycles. Simulations of the four-slot mechanism found in [1] have shown that it usually has worse data loss and re-reading rates than a mechanism of the type

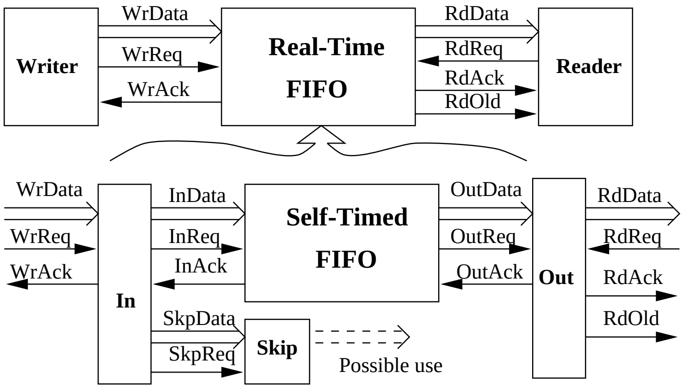

Figure 1: Structure of real-time FIFO

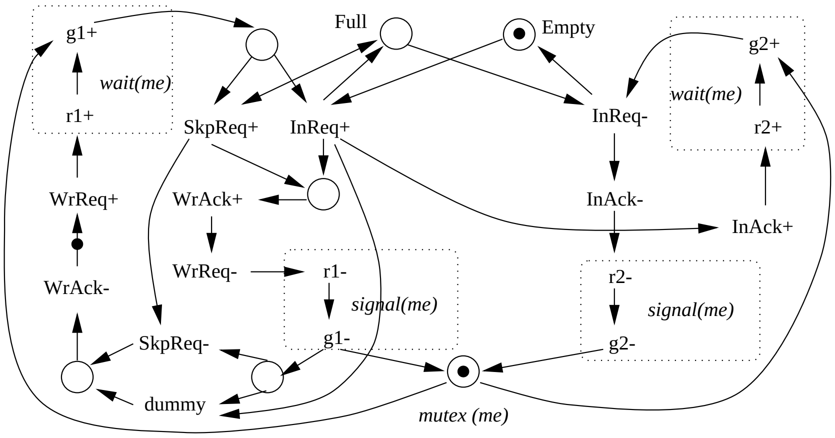

Figure 2: STG model of the In block with mutex actions

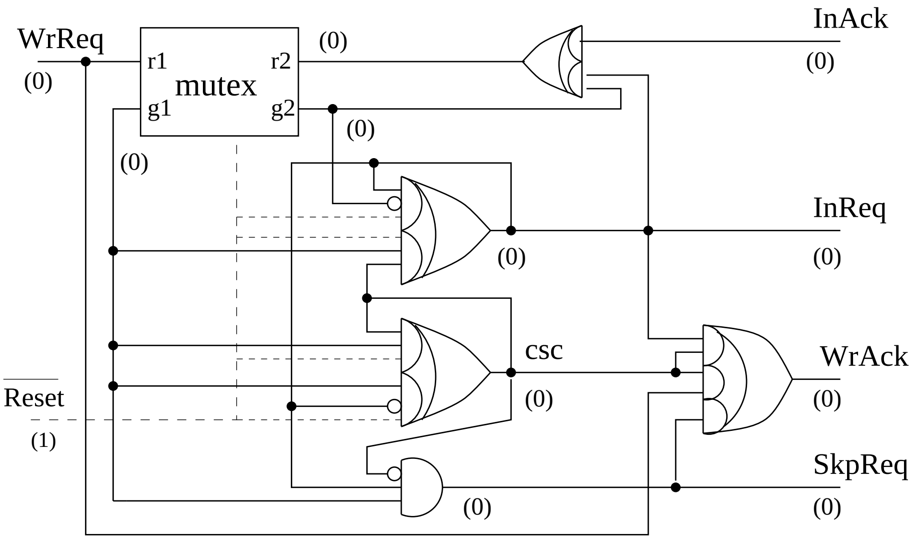

Figure 3: Circuit implementation of the In interface.

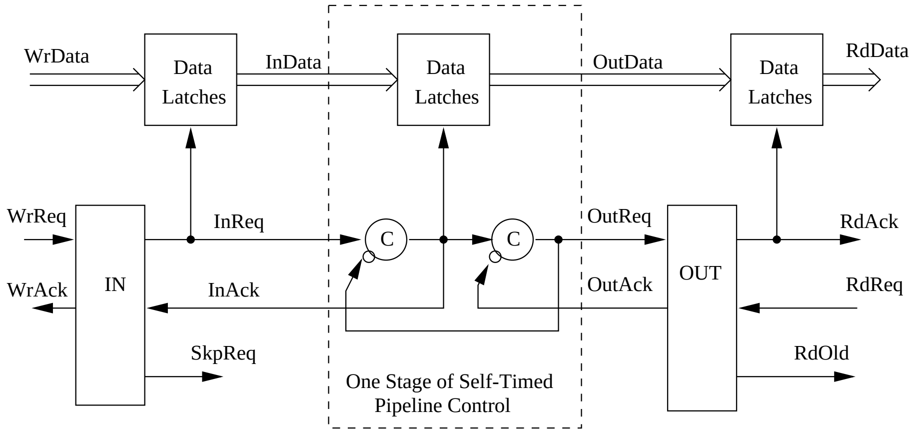

Figure 4: Interconnection with data path

presented here with a two-stage FIFO.

3 Conclusion

The proposed method of asynchronous communication and its circuit implementation offer the following advantages for the "profile data" applications:

- (i) simplicity, both in its actual description and its intuitive data coherence;
- (ii) data sequencing, guaranteed by definition for the use of FIFO;
- (iii) reduced miss (skipping and re-reading) rate,
- (iv) flexibility and adjustability to suit known write/read ratios (by FIFO size "tuning").

Our FIFO buffer with non-blocking Write-Read interface can be seen as an alternative or complementary solution to the four-slot mechnanism [1], which was aimed at the freshness issue. It should be pointed out that the use of a pipeline (i.e., linear) architecture for the self-timed FIFO is chosen only for simplicity. The pipeline register is known to have a simple control logic structure, which also renders high data throughput. At the same time, with a pipeline one loses in latency, especially if the size of the buffer has to be large. Alternative solutions can be based on a parallel architecture with demultiplexor and multiplexor or on a RAM with an additional control structure with up-down counters. All these approaches are known to allow asynchronous circuit implementation (cf. [8]).

Acknowledgement. This work was supported by EPSRC (project COMFORT, GR/L93775). The authors would like to thank E. Campbell, H. Simpson, I. Clark and A. Davies for helpful discussions.

References

[1] H.R. Simpson. Four-slot fully asynchronous communication mechanism. *Proc IEE*, Vol. 137, Pt.E, No.1, Jan. 1990, pp.17-30.

[2] I.E. Sutherland. Micropipelines (The Turing Award Lecture). *Communications of ACM*, Vol.32, No.6, January 1989, pp.226-231.

[3] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. *IEICE Trans. Inf. and Syst.*, E80-D(3):315–325, March 1997.

[4] T. Murata. Petri nets: Properties, analysis and applications. *Proc. IEEE*, Vol. 77, No. 4, April 1989, pp. 541-580.

[5] A.V. Yakovlev, D.J. Kinniment, and F. Xia A FIFO buffer with real-time interface. TR. no. 649 Department of Computing Science, University of Newcastle upon Tyne, 1998.

[6] C. L. Seitz. Ideas about arbiters. Lambda, Vol. 1, First Quarter 1980, pp. 10-14.

[7] D.J. Kinniment, A.V. Yakovlev, and B. Gao. Metastable behaviour in arbiter circuits. Tech. Rep. Series 604, Department of Computing Science, University of Newcastle upon Tyne, 1997.

[8] A.V. Yakovlev, A.M. Koelmans, and L. Lavagno. High-level modelling and design of asynchronous interface logic. *IEEE Design and Test of Computers*, Vol. 12, No. 1, Spring 1995, pp. 32-40.