A FIFO buffer with non-blocking interface

A. Yakovlev, D.J. Kinniment, F. Xia and A.M. Koelmans

Department of Computing Science, University of Newcastle upon Tyne,
Claremont Tower, Claremont Road, Newcastle upon Tyne, NE1 7RU, UK
Email contact: Alex.Yakovlev@ncl.ac.uk

Abstract. The design of a FIFO buffer supporting synchronisation-free, non-blocking, reading and writing operations is presented. The circuit consists of a standard self-timed FIFO surrounded by two interface blocks, In and Out, which together provide temporal independence between the reader and writer processes. These blocks are synthesised from their Signal Transition Graph (STG) specifications using an asynchronous circuit synthesis tool, `Petrify`. The proposed solution is compared with "slot" mechanisms. It is shown to be more appropriate for "profile data" applications, such as video or audio transmissions.

Introduction

Real time systems often need to provide data communication between subsystems with independent timing of the 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 the ability to accommodate subsystems with different clocks on the same chip.

For such applications, any scheme involving any temporal blocking between a pair of Reader and Writer processes may not be acceptable. Here it is assumed that shared memory is used between Reader and Writer, in the form of Fig 1 in [1]. For finite systems of this type, two necessary consequences of providing temporal independence for Reader and Writer (or completely "de-coupling" them temporally) 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 Reader must be allowed to re-read old data it has already obtained in a previous cycle of operation.

Such asynchronous communication mechanisms (ACMs), however, should possess such properties as *data coherence* and *data freshness*. The former implies that Reader should not access data that is being written and Writer should not attempt writing to data that is being read. The latter implies that Reader should ideally be able to obtain the newest possible data item available, given the physical limitations of the device being designed.

Another important property of ACMs is *data sequencing*, which implies that Reader should ideally at no time obtain an item of data older than the item obtained during its previous cycle.

Most of the ACMs published to date involve multiple data slots, which are organised in a conceptually parallel/transverse fashion. This implies that each data slot may be potentially accessible to either Reader or Writer at any time. Potential conflicts on any single slot are normally resolved by the use of control variables. Examples of these types of mechanisms can be found in [2]. They are referred to as *slot mechanisms* here.

The slot mechanisms, because of the transverse organisation of slots, do not provide the design flexibility to influence the data loss, re-reading, and sequencing properties of implementations. The designers are left claiming that data loss and data re-reading are "inevitable consequences of synchronisation-free data communication" and leaving data sequencing to fortune. As a result, certain designs are able to maintain data sequencing but others, obtained using similar methods, not. To date there is no clear idea as to why this is the case [3].

This has resulted in the designers' targeting "reference data" type applications for the slot mechanisms.

In order to broaden the application field of non-blocking ACMs, a design is presented here which provides some means of adjusting 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 into the ACM, which would be more suitable for "profile data" applications such as speech and image data. This type of data is more context sensitive than "reference data". On the other hand, absolute minimum latency is not always required in transmission.

FIFO buffer design

The classical bounded self-timed buffer [4] helps to smooth out any temporary differences in the rate between data writing and reading. However, if the average speeds of Reader and Writer differ it would eventually start blocking (obliging waiting) the activity of the faster party.

A FIFO buffer has been designed 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 so that Reader may use the previously read data item. These are the data loss and re-reading provisions to ensure non-blocking.

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 block show (similar conditions apply for the Out block).

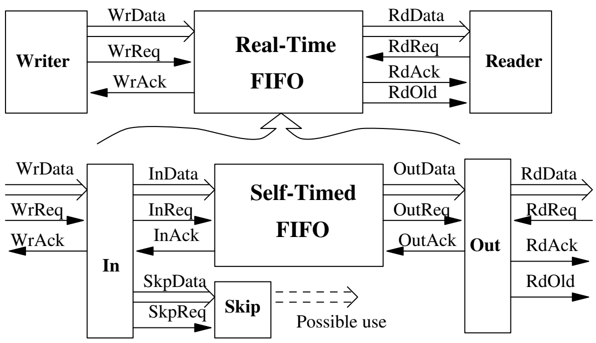

Figure 1 Structure of real-time FIFO

Real-time 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 block and perhaps the propagation delay of the first stage of the self timed FIFO when that stage is empty.

The operation of the In and Out blocks can be defined using STGs, which are a special interpretation of Petri nets where events are associated with rising and falling signal transitions. In this way STGs are like timing diagrams. The STG model of the In block is shown in Figure 2.

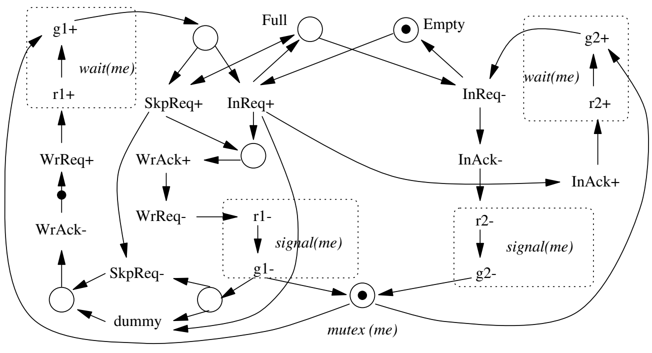

Figure 2 STG model of the In block with mutex actions.

This model includes special mutex actions to protect the critical sections which change/test the state of the full/empty flag in the circuit. This mutex is implemented through a standard 2-way arbiter [5]. The circuit synthesised with the `Petrify` tool [6] is shown in Figure 3. The design of the Out block is described in [7].

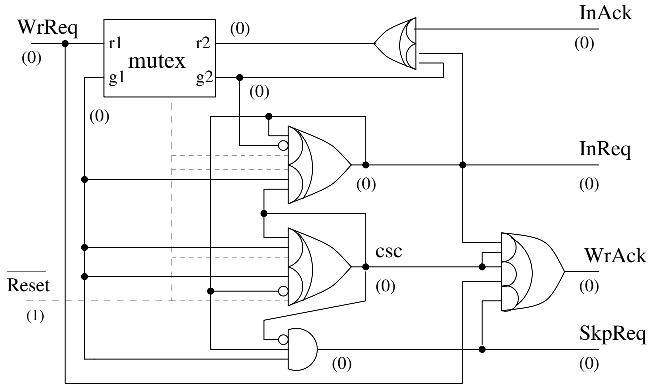

Figure 3 Circuit implementation of the In block.

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 latches in the data path are edge-triggered.

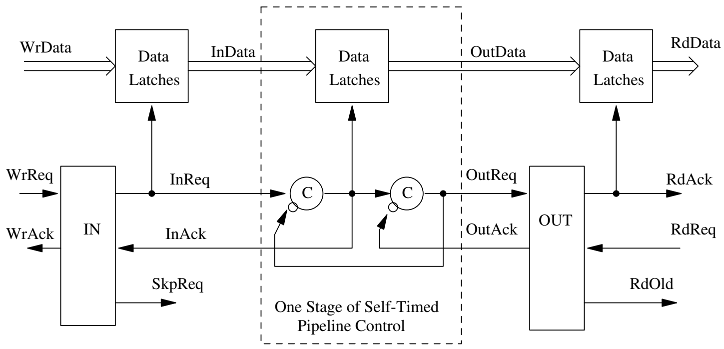

Figure 4 Interconnection with data path.

The worst case signal transition path for Writer (similar analysis for Reader is in [7]) corresponds to the situation when the WrReq+ transition occurs concurrently with the arrival of InAck+ from the first stage of the FIFO (after the previous data item 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 "FIFO side" to request (and hence win) arbitration more than once without having executed at lease one cycle of the (WrReq, WrAck) handshake. Thus, Writer's process can only be delayed at most for one FIFO handshake half-cycle, between InReq- and InAck-. This period is of a predictable length because, as can be seen from Figure 4, the resetting phase of the stage does not depend on the state of the subsequent stage. 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 r1+ and r2+. In practice, the bound of this delay can be estimated conservatively with extremely high probability [8]. In any case, the bound of metastability resolution time for a bistable must be taken into account in any solution of synchronisation-free communications. 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 condition.

Analysis in [7] shows that this ACM 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 [2] have shown that it usually has worse data loss and re-reading rates than an ACM of the type presented here if the latter has at least a two-stage FIFO.

Conclusion and acknowledgement

The proposed ACM and its circuit implementation offer the following advantages for "profile data" applications:

1. simplicity, both in its actual description and its intuitive data coherence;
2. data sequencing, guaranteed by definition for the use of a FIFO;
3. a variable measure within the ACM by which the data loss and re-reading rates can be adjusted to suite known Writer and Reader statistical information;
4. components which have distinct functions relating to the properties (In and Out used for timing independence and FIFO used to adjust data loss and re-reading rates).

It can be seen as an alternative or complementary non-blocking ACM to the slot mechanisms when absolute data freshness is not required.

It should be pointed out that the use of a linear pipeline structure for the self-timed FIFO is purely for convenience. Alternative solutions may be based on parallel architecture with demultiplexor and multiplexor or a RAM with an additional control structure with up-down counters. All these approaches are known to allow asynchronous circuit implementations [9]. These may have smaller latency than a linear FIFO.

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

References

1 Xia, F. and Liu, B., "Quantitative investigation of certain properties of slot mechanisms", elsewhere in this Forum.

2 Simpson, H.R., "Four-slot fully asynchronous communication mechanism", IEE Proceedings, Vol. 137, Pt. E, No. 1, pp.17-30, January 1990.

3 Simpson, H.R., "New algorithms for asynchronous communication" and "Role model analysis of an asynchronous communication mechanism", IEE Proceedings on Computing and Digital Techniques, Vol. 144, No. 4, pp.227-240, July 1997.

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

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

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

7 Yakovlev, A., Kinnement, D.J. and Xia, F., "A FIFO buffer with real-time interface", Technical Report, TR649, Department of Computing Science, University of Newcastle upon Tyne, 1998.

8 Kinniment, D.J., Yakovlev, A. and Gao, B., "Metastable behaviour in arbiter circuits", Technical Report, TR604, Department of Computing Science, University of Newcastle upon Tyne, 1997.

9 Yakovlev, A., Koelmans, A. and Lavagno, L., "High level modelling and design of asynchronous interface logic", IEEE Design and Test of Computers, Vol. 12, No. 1, pp.32-40, Spring 1995.