WORKSHOPS IN COMPUTING

Series edited by Professor C.J. van Rijsbergen


G. Birtwistle and A. Davis (Eds)

# Asynchronous Digital Circuit Design

Springer

G. Birtwistle and A. Davis (Eds)

# Asynchronous Digital Circuit Design



Published in collaboration with the
British Computer Society


Springer

London Berlin Heidelberg New York
Paris Tokyo Hong Kong
Barcelona Budapest

Graham Birtwistle, BSc, PhD, DSc
Department of Computer Science,
University of Calgary, 2500 University Drive,
Calgary, Alberta, T2N 1N4, Canada

Alan Davis, BSEE, PhD
Department of Computer Science,
University of Utah, Salt Lake City,
UT 84112, USA

ISBN 3-540-19901-2 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-19901-2 Springer-Verlag New York Berlin Heidelberg

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

Library of Congress Cataloging-in-Publication Data
Asynchronous digital circuit design / G. Birtwistle and A. Davis. eds.
p. cm. - (Workshops in computing)
Based on a workshop held in Banff, Alberta, August 28-September 3, 1993.
"Published in collaboration with the British Computer Society."
Includes bibliographical references and index.
ISBN 3-540-19901-2
1. Digital integrated circuits-Design and construction-Data processing.
2. Asynchronous circuits-Design and contruction-Data processing. 3. Computer-aided design. I. Birtwistle, G.M. (Graham Mark), 1946- .
II. Davies A. (Alan Lynn), 1946- . III. British Computer Society. IV. Series.
TK7874.65.A88 1995 95-1009
621.39'5-dc20 CIP



Typesetting: Camera ready by contributors
Printed by Athenæum Press Ltd., Gateshead
34/3830-543210 Printed on acid-free paper

WORKSHOPS IN COMPUTING
Series edited by C. J. van Rijsbergen

## *Also in this series*

**Logic Program Synthesis and Transformation**
Proceedings of LOPSTR 93, International Workshop on Logic Program Synthesis and Transformation, Louvain-la-Neuve, Belgium, 7–9 July 1993
Yves Deville (Ed.)

**Database Programming Languages (DBPL-4)**
Proceedings of the Fourth International Workshop on Database Programming Languages – Object Models and Languages, Manhattan, New York City, USA, 30 August–1 September 1993
Catriel Beeri, Atsushi Ohori and Dennis E. Shasha (Eds)

**Music Education: An Artificial Intelligence Approach,** Proceedings of a Workshop held as part of AI-ED 93, World Conference on Artificial Intelligence in Education, Edinburgh, Scotland, 25 August 1993
Matt Smith, Alan Smaill and Geraint A. Wiggins (Eds)

**Rules in Database Systems**
Proceedings of the 1st International Workshop on Rules in Database Systems, Edinburgh, Scotland, 30 August–1 September 1993
Norman W. Paton and M. Howard Williams (Eds)

**Semantics of Specification Languages (SoSL)**
Proceedings of the International Workshop on Semantics of Specification Languages, Utrecht, The Netherlands, 25–27 October 1993
D.J. Andrews, J.F. Groote and C.A. Middelburg (Eds)

**Security for Object-Oriented Systems**
Proceedings of the OOPSLA-93 Conference Workshop on Security for Object-Oriented Systems, Washington DC, USA, 26 September 1993
B. Thuraisingham, R. Sandhu and T.C. Ting (Eds)

**Functional Programming, Glasgow 1993**
Proceedings of the 1993 Glasgow Workshop on Functional Programming, Ayr, Scotland, 5–7 July 1993
John T. O'Donnell and Kevin Hammond (Eds)

**Z User Workshop, Cambridge 1994**
Proceedings of the Eighth Z User Meeting, Cambridge, 29–30 June 1994
J.P. Bowen and J.A. Hall (Eds)

**6th Refinement Workshop**
Proceedings of the 6th Refinement Workshop, organised by BCS-FACS, London, 5–7 January 1994
David Till (Ed.)

**Incompleteness and Uncertainty in Information Systems**
Proceedings of the SOFTEKS Workshop on Incompleteness and Uncertainty in Information Systems, Concordia University, Montreal, Canada, 8–9 October 1993
V.S. Alagar, S. Bergler and F.Q. Dong (Eds)

**Rough Sets, Fuzzy Sets and Knowledge Discovery**
Proceedings of the International Workshop on Rough Sets and Knowledge Discovery (RSKD'93), Banff, Alberta, Canada, 12–15 October 1993
Wojciech P. Ziarko (Ed.)

**Algebra of Communicating Processes**
Proceeedings of ACP94, the First Workshop on the Algebra of Communicating Processes, Utrecht, The Netherlands, 16–17 May 1994
A. Ponse, C. Verhoef and S.F.M. van Vlijmen (Eds)

**Interfaces to Database Systems (IDS94)**
Proceedings of the Second International Workshop on Interfaces to Database Systems, Lancaster University, 13–15 July 1994
Pete Sawyer (Ed.)

**Persistent Object Systems**
Proceedings of the Sixth International Workshop on Persistent Object Systems, Tarascon, Provence, France, 5–9 September 1994
Malcolm Atkinson, David Maier and Véronique Benzaken (Eds)

**Functional Programming, Glasgow 1994**
Proceedings of the 1994 Glasgow Workshop on Functional Programming, Ayr, Scotland, 12–14 September 1994
Kevin Hammond, David N. Turner and Patrick M. Sansom (Eds)

**East/West Database Workshop**
Proceedings of the Second International East/West Database Workshop, Klagenfurt, Austria, 25–28 September 1994
J. Eder and L.A. Kalinichenko (Eds)

*continued on back page...*

# Preface

The contents of this book are an expanded and thorough treatment of a set of presentations made at a workshop held in Banff, Alberta, 28 August – 3 September 1993 by leading practitioners in asynchronous hardware design. The papers cover a wide range of current practice from practical design, through silicon compilation, to the applications of formal specifications.

Jo Ebergen, John Segers and Igor Benko (Waterloo) describe work on the formal specification of asynchronous circuits in a CSP-like notation which they then go on to analyze for delay, safety, progress, and performance issues.

Al Davis (Utah) describes a set of automatic asynchronous design synthesis tools that were used to implement a large, compact and fast industrial design (a message passing post-office). These tools have been integrated into an existing commercial VLSI CAD framework. The tools, their usage, algorithms, and interfaces are presented.

Kees van Berkel (Philips) and Martin Rem (Eindhoven) view VLSI design as a programming activity using the CSP-based programming language Tangram. They show how Tangram descriptions can be compiled into handshake circuits and thence into layout.

Steve Furber (Manchester) describes and motivates the design of a working asynchronous implementation of the ARM microprocessor developed at Manchester University. The design is based upon Sutherland's micropipelines. The architecture is described in detail and an evaluation of the first silicon is presented.

The collection is rounded out by a paper by Steven Nowick (Columbia) and Al Davis which gives a state-of-the-art survey of asynchronous hardware design. This paper was not presented at the workshop.

Alain Martin (CalTech) also lectured at the workshop but was precluded from contributing to this volume by pressures of work.

The workshop was held at the Banff Conference Center and ran, as ever, as though by clockwork. It is a pleasure to record our thanks to their well-organised, and ever helpful and friendly staff.

Graham Birtwistle
Calgary

Al Davis
Utah

## *Published in 1990–92*

**AI and Cognitive Science '89,** Dublin City University, Eire, 14–15 September 1989
Alan F. Smeaton and Gabriel McDermott (Eds)

**Specification and Verification of Concurrent Systems,** University of Stirling, Scotland, 6–8 July 1988
C. Rattray (Ed.)

**Semantics for Concurrency,** Proceedings of the International BCS-FACS Workshop, Sponsored by Logic for IT (S.E.R.C.), University of Leicester, UK, 23–25 July 1990
M. Z. Kwiatkowska, M. W. Shields and R. M. Thomas (Eds)

**Functional Programming, Glasgow 1989**
Proceedings of the 1989 Glasgow Workshop, Fraserburgh, Scotland, 21–23 August 1989
Kei Davis and John Hughes (Eds)

**Persistent Object Systems,** Proceedings of the Third International Workshop, Newcastle, Australia, 10–13 January 1989
John Rosenberg and David Koch (Eds)

**Z User Workshop, Oxford 1989,** Proceedings of the Fourth Annual Z User Meeting, Oxford, 15 December 1989
J. E. Nicholls (Ed.)

**Formal Methods for Trustworthy Computer Systems (FM89),** Halifax, Canada, 23–27 July 1989
Dan Craigen (Editor) and Karen Summerskill (Assistant Editor)

**Security and Persistence,** Proceedings of the International Workshop on Computer Architectures to Support Security and Persistence of Information, Bremen, West Germany, 8–11 May 1990
John Rosenberg and J. Leslie Keedy (Eds)

**Women into Computing: Selected Papers 1988–1990**
Gillian Lovegrove and Barbara Segal (Eds)

**3rd Refinement Workshop** (organised by BCS-FACS, and sponsored by IBM UK Laboratories, Hursley Park and the Programming Research Group, University of Oxford), Hursley Park, 9–11 January 1990
Carroll Morgan and J. C. P. Woodcock (Eds)

**Designing Correct Circuits,** Workshop jointly organised by the Universities of Oxford and Glasgow, Oxford, 26–28 September 1990
Geraint Jones and Mary Sheeran (Eds)

**Functional Programming, Glasgow 1990**
Proceedings of the 1990 Glasgow Workshop on Functional Programming, Ullapool, Scotland, 13–15 August 1990
Simon L. Peyton Jones, Graham Hutton and Carsten Kehler Holst (Eds)

**4th Refinement Workshop,** Proceedings of the 4th Refinement Workshop, organised by BCS-FACS, Cambridge, 9–11 January 1991
Joseph M. Morris and Roger C. Shaw (Eds)

**AI and Cognitive Science '90,** University of Ulster at Jordanstown, 20–21 September 1990
Michael F. McTear and Norman Creaney (Eds)

**Software Re-use, Utrecht 1989,** Proceedings of the Software Re-use Workshop, Utrecht, The Netherlands, 23–24 November 1989
Liesbeth Dusink and Patrick Hall (Eds)

**Z User Workshop, 1990,** Proceedings of the Fifth Annual Z User Meeting, Oxford, 17–18 December 1990
J.E. Nicholls (Ed.)

**IV Higher Order Workshop, Banff 1990**
Proceedings of the IV Higher Order Workshop, Banff, Alberta, Canada, 10–14 September 1990
Graham Birtwistle (Ed.)

**ALPUK91,** Proceedings of the 3rd UK Annual Conference on Logic Programming, Edinburgh, 10–12 April 1991
Geraint A.Wiggins, Chris Mellish and Tim Duncan (Eds)

**Specifications of Database Systems**
International Workshop on Specifications of Database Systems, Glasgow, 3–5 July 1991
David J. Harper and Moira C. Norrie (Eds)

**7th UK Computer and Telecommunications Performance Engineering Workshop**
Edinburgh, 22–23 July 1991
J. Hillston, P.J.B. King and R.J. Pooley (Eds)

**Logic Program Synthesis and Transformation**
Proceedings of LOPSTR 91, International Workshop on Logic Program Synthesis and Transformation, University of Manchester, 4–5 July 1991
T.P. Clement and K.-K. Lau (Eds)

**Declarative Programming, Sasbachwalden 1991**
PHOENIX Seminar and Workshop on Declarative Programming, Sasbachwalden, Black Forest, Germany, 18–22 November 1991
John Darlington and Roland Dietrich (Eds)

**Building Interactive Systems: Architectures and Tools**
Philip Gray and Roger Took (Eds)

**Functional Programming, Glasgow 1991**
Proceedings of the 1991 Glasgow Workshop on Functional Programming, Portree, Isle of Skye, 12–14 August 1991
Rogardt Heldal, Carsten Kehler Holst and Philip Wadler (Eds)

**Object Orientation in Z**
Susan Stepney, Rosalind Barden and David Cooper (Eds)

**Code Generation – Concepts, Tools, Techniques**
Proceedings of the International Workshop on Code Generation, Dagstuhl, Germany, 20–24 May 1991
Robert Giegerich and Susan L. Graham (Eds)

**Z User Workshop, York 1991,** Proceedings of the Sixth Annual Z User Meeting, York, 16–17 December 1991
J.E. Nicholls (Ed.)

**Formal Aspects of Measurement**
Proceedings of the BCS-FACS Workshop on Formal Aspects of Measurement, South Bank University, London, 5 May 1991
Tim Denvir, Ros Herman and R.W. Whitty (Eds)

**AI and Cognitive Science '91** University College, Cork, 19–20 September 1991
Humphrey Sorensen (Ed.)

**5th Refinement Workshop,** Proceedings of the 5th Refinement Workshop, organised by BCS-FACS, London, 8–10 January 1992
Cliff B. Jones, Roger C. Shaw and Tim Denvir (Eds)

**Algebraic Methodology and Software Technology (AMAST'91)**
Proceedings of the Second International Conference on Algebraic Methodology and Software Technology, Iowa City, USA, 22–25 May 1991
M. Nivat, C. Rattray, T. Rus and G. Scollo (Eds)

**ALPUK92,** Proceedings of the 4th UK Conference on Logic Programming, London, 30 March – 1 April 1992
Krysia Broda (Ed.)

**Logic Program Synthesis and Transformation**
Proceedings of LOPSTR 92, International Workshop on Logic Program Synthesis and Transformation, University of Manchester, 2–3 July 1992
Kung-Kiu Lau and Tim Clement (Eds)

**NAPAW 92,** Proceedings of the First North American Process Algebra Workshop, Stony Brook, New York, USA, 28 August 1992
S. Purushothaman and Amy Zwarico (Eds)

**First International Workshop on Larch**
Proceedings of the First International Workshop on Larch, Dedham, Massachusetts, USA, 13–15 July 1992
Ursula Martin and Jeannette M. Wing (Eds)

**Persistent Object Systems**
Proceedings of the Fifth International Workshop on Persistent Object Systems, San Miniato (Pisa), Italy, 1–4 September 1992
Antonio Albano and Ron Morrison (Eds)

**Formal Methods in Databases and Software Engineering,** Proceedings of the Workshop on Formal Methods in Databases and Software Engineering, Montreal, Canada, 15–16 May 1992
V.S. Alagar, Laks V.S. Lakshmanan and F. Sadri (Eds)

**Modelling Database Dynamics**
Selected Papers from the Fourth International Workshop on Foundations of Models and Languages for Data and Objects, Volkse, Germany, 19–22 October 1992
Udo W. Lipeck and Bernhard Thalheim (Eds)

**14th Information Retrieval Colloquium**
Proceedings of the BCS 14th Information Retrieval Colloquium, University of Lancaster, 13–14 April 1992
Tony McEnery and Chris Paice (Eds)

**Functional Programming, Glasgow 1992**
Proceedings of the 1992 Glasgow Workshop on Functional Programming, Ayr, Scotland, 6–8 July 1992
John Launchbury and Patrick Sansom (Eds)

**Z User Workshop, London 1992**
Proceedings of the Seventh Annual Z User Meeting, London, 14–15 December 1992
J.P. Bowen and J.E. Nicholls (Eds)

**Interfaces to Database Systems (IDS92)**
Proceedings of the First International Workshop on Interfaces to Database Systems, Glasgow, 1–3 July 1992
Richard Cooper (Ed.)

**AI and Cognitive Science '92**
University of Limerick, 10–11 September 1992
Kevin Ryan and Richard F.E. Sutcliffe (Eds)

**Theory and Formal Methods 1993**
Proceedings of the First Imperial College Department of Computing Workshop on Theory and Formal Methods, Isle of Thorns Conference Centre, Chelwood Gate, Sussex, UK, 29–31 March 1993
Geoffrey Burn, Simon Gay and Mark Ryan (Eds)

**Algebraic Methodology and Software Technology (AMAST'93)**
Proceedings of the Third International Conference on Algebraic Methodology and Software Technology, University of Twente, Enschede, The Netherlands, 21–25 June 1993
M. Nivat, C. Rattray, T. Rus and G. Scollo (Eds)

# Contents

# Asynchronous Circuit Design: Motivation, Background, & Methods

Al Davis

Department of Computer Science, University of Utah
Salt Lake City, UT 84112, USA

Steven M. Nowick

Department of Computer Science, Columbia University
New York, NY 10027, USA

**Abstract**

The purpose of this book is to present a current view of the state of the art for the field of asynchronous circuit design and analysis which was the topic of a workshop in Banff in the fall of 1993. Asynchronous circuits have been studied in one form or another since the early 1950's [64] when the focus was primarily on mechanical relay circuits and when the differences between the asynchronous and clocked circuit design styles were somewhat indistinct. A number of theoretical issues were studied in detail by Muller and Bartky as early as 1956 [92]. Since then, the field of asynchronous circuits has gone through a number of high-interest cycles. In the last 5 years there has been an unprecedented level of interest in both academic and industrial settings [56]. This historical trend continues today with the majority of the current research effort focused more on theory than on practice. Nonetheless, the advance of practical asynchronous circuit design techniques also has an unusual level of interest. The work presented at the Banff workshop was concerned more with practice than theory and provided a reasonable coverage of the current approaches to asynchronous circuit design. Similarly this chapter will primarily focus on practical design issues. Prior to introducing the four chapters which follow, we present an introduction to the basic concepts and motivations behind asynchronous circuit design. This will hopefully enable those not already familiar with asynchronous circuit design to better understand the subsequent chapters.

## 1 Motivation and Basic Concepts

Circuit design styles can be classified into two major categories, namely synchronous and asynchronous. It is worthwhile to note that neither is independent of the other and that there are many designs that have been produced using a hybrid design style which mixes aspects of both categories. There is also considerable debate within the asynchronous circuit community as to what constitutes a *pure* asynchronous circuit. It may be difficult to understand the motivation for asynchronous circuit techniques when the bulk of commercial practice and considerable experience, artifact, and momentum exists for the synchronous circuit design style. For some, the motivation to pursue the study of asynchronous circuits is based on the simple fact that they are different. Others find that asynchronous circuits have a particular modular elegance that

is amenable to theoretical analysis. However, for those interested in the practical aspects of asynchronous circuit design, the motivation often comes from some concern with the basic nature of synchronous circuits.

Of common concern are the cost issues associated with the global, periodic, and common clock that is the temporal basis for all purely synchronous circuits. The *fixed* clock period of synchronous circuits is chosen as a result of worst-case timing analysis. It is not adaptive and therefore does not take advantage of average- or even best-case computational situations. Asynchronous circuit proponents view this as an opportunity to achieve increased performance since asynchronous methods are inherently adaptive. Arithmetic circuits are a good example in this regard. Arithmetic circuit performance is typically dominated by the propagation delay of carry or borrow signals. The worst-case propagation situation rarely occurs, yet synchronous arithmetic circuits must be clocked in a manner that accommodates this rare worst-case condition. Some asynchronous circuit designers have made the mistake of generalizing this observation into a view that the inherent adaptivity of asynchronous circuits implies that they are capable of achieving higher performance in general, but this is not necessarily the case.

All asynchronous circuits have additional operational constraints when compared to their synchronous counterparts. All forms of asynchronous circuits are concerned with providing hazard or *glitch* free outputs under some timing model. In order to achieve this behavior, the asynchronous circuit will often contain more gates than a functionally equivalent synchronous circuit. Therefore in terms of the number of basic components, asynchronous circuits are often somewhat larger than synchronous circuits. More gates implies more wires, and this often results in slower rather than faster circuit latencies. Furthermore in order to achieve their inherently adaptive nature, asynchronous circuits must generate their own control signals such as a request and an acknowledge signal. The acknowledge signal indicates completion of a previously requested action. In synchronous circuits much of this type of control signaling is implicit in the common clock signal. The generation of these additional control signals further exacerbates the complexity of asynchronous circuits, and may lead to a further performance degradation.

The adaptive potential remains where the worst-case situation is rare and when the difference between the worst-case and average-case latencies is significant. However, synchronous circuit designers are also well aware of this situation and take considerable care to create a clock model and circuit structure that can take advantage of these differences. The most notable example of this tactic is in the finely-grained pipeline structures of modern floating point units. Yet, for very large circuits, such as microprocessors, balancing all the timing constraints of a large computational space so that there is little difference between the worst and average case timing models is a difficult task. The work by Mark Dean on the STRiP processor [39] provides an interesting example. Dean showed that even a well-balanced and well-designed processor such as the MIPS-X CPU could be sped up if the instruction set were split into three classes, and the clock period adjusted appropriately to match the

temporal needs of each class.

Dean also demonstrated that an even greater performance enhancement could be achieved due to the tighter margins which are possible with adaptive clocking. Synchronous systems usually rely on an externally-generated clock signal which is distributed as the common timing reference to all of the system components. The speed at which integrated circuits operate varies with the circuit fabrication process, and fluctuations in operating temperature and supply voltage. In order to achieve a reasonable shield against these variables, the clock period is extended by a certain *margin.* In current practice, these margins are often 100% or more for high-speed systems. Adaptive clocking cannot be generated externally, and therefore must be provided internally to each device. The fact that the clock generator is affected by the same process, temperature, and supply variations as the rest of the chip permits the safety margin to be reduced significantly.

Clock distribution is becoming an increasingly costly component of large modern designs. Today's microprocessors contain over two million transistors and their clock rates are around 200 MHz. The clock period is determined by adding the worst case propagation delay, the margin, and the maximum clock skew. Clock skew is simply the maximum difference in the clock arrival as seen by all clocked points in the circuit. The latency of the clock pulse to the reception points is not a concern. With today's large VLSI circuits exceeding 15 mm per side, several nanoseconds of skew is easily possible. However with a 5 nanosecond clock period, several nanoseconds of skew is a disaster. Clock distribution and deskewing methods are abundant but they share the common characteristic of being extremely expensive. A common method is to distribute the clock via a balanced H-tree configuration [6] with amplifying buffers placed at the fanout points. The problem with this approach is that as more buffers are added to a clock path, larger skew results. The designers of the DEC Alpha CPU [119] took the opposite approach. The Alpha contains 1.68 million transistors and is fabricated in a .75 micron, 3.3 volt CMOS process. Even with three layers of metal, the chip is 16.8 mm by 13.9 mm. In order to keep clock skew to within 300 picoseconds, the Alpha's designers localized the clock buffering to minimize process induced variations and therefore the skew induced by the buffers. Details of the method can be found in [45] but the result is a clock driver circuit that occupies about 10% of the chip area, and consumes over 40% of the 30 watts of power dissipated by the chip. 19 $mm^2$ of area and over 12 watts of power is a very high price to pay for keeping the skew under control. Clearly this technique will be difficult to extend to a domain where circuit speeds and transistor counts double.

A similar skew problem exists for circuit boards as well as chips. The literature contains an abundance of methods for deskewing clocks [2, 26] on a board but most of them are costly in either area or complexity, and some will probably not be robust enough for use in commercial circuits. An interesting example is the Monarch [110] processor chip which used active signal selection on each input pad. In this instance, a five slot delay line was used to skew signals to match the clock skew. The appropriate tap in the delay line was selected

based on analyzing the clock vs. the incoming signal. While the technique did work, its cost and complexity are probably more instructive in a pathological sense. The bottom line is that clock management is a difficult problem and solving it in today's high-speed, highly complex designs is costly. Asynchronous circuit proponents advocate a simple solution, namely throw away the whole concept of a global clock. This is not a free solution since global absolute timing must be replaced with the relative and sequential mechanisms which lie at the heart of asynchronous circuit signaling protocols. Chuck Seitz wrote an excellent introduction to this general topic in his chapter on *System Timing* in the classic VLSI book by Mead and Conway [85]. The next section of this treatise presents some of the more commonly used protocols and terminology.

Another common motivation for pursuing the asynchronous circuit option is the quest for low-power circuit operation. The consumer market's hunger for powerful yet portable digital systems which run on lightweight battery packs is growing at a rapid rate. Hence there is a strong commercial interest in low-power design methods which extend the operational life of a particular battery technology. CMOS circuits have a particular appeal since they consume negligible power when they are idle. This would not be true however if the clock of a synchronous circuit were to continue running. Therefore, low power synchronous circuits usually involve some method of shutting down the clock to most of the system. The exception is the subcomponent that must monitor the environment for the next call to action. However these techniques often result in increased clock skew problems. Asynchronous circuits have the advantage that they go into idle mode for free since, by nature, when there is nothing to do there are no transitions on any wire in the circuit. Another advantage is that even for an active system, only the subsystems that are required for the computation at hand will dissipate any power. Researchers such as Kees van Berkel [133] and Steve Furber [51] are pursuing asynchronous circuit designs where they attempt to exploit this feature.

The final motivation is based on the inherent ease of composing asynchronous subsystems into larger asynchronous systems. While there is still room for doubt about whether asynchronous circuits can achieve their potential advantages in terms of higher performance or lower power operation than synchronous circuits, there is little doubt that asynchronous circuits do have a definite advantage with respect to composability. Asynchronous circuits are functional modules in that they contain both their timing and data requirements *explicitly* in their interfaces. In a sense they "keep time for themselves", hence the term self-timed circuits. Synchronous circuit modules contain only data requirements in their interfaces and share *the clock.* However, important temporal issues such as when data must be valid to avoid set-up and hold time violations between modules are *implicit* at best. Composing asynchronous modules is almost trivial. If the interfaces match and observe the same signaling protocol then they can simply be connected. More detailed knowledge of module internals is required before synchronous subsystems can be connected.

The problem of combining synchronous systems is exacerbated when each module has a separate clock, each running at a different frequency. The effects

of this problem are numerous and usually involve some variant of metastability failure [25]. It is commonly accepted, although not definitively proven to the authors' knowledge, that it is impossible to build a perfect synchronizer. Many of the subsystems in today's computers run on clocks which are not synchronized with the CPU. A good example is the I/O subsystem. In this case, techniques must be used which trade increased latency for more reliable synchronization. The reliability is adjusted to meet the MTBF (Mean Time Before Failure) requirements of the system, and the resulting decreased performance is simply viewed as the price that must be paid for the required reliability.

The ease of composing asynchronous subsystems is a tremendous advantage. It allows components from previous designs to be reused, it permits modification of slower components to permit incremental performance improvements without impacting the overall design, and facilitates behavioral analysis by formal methods. However, asynchronous circuits are presently not the mainstay of commercial practice. The definite advantage of composability is not a strong enough factor to counter the significant synchronous circuit momentum, and the promises of improved performance and decreased power consumption remain to be generally realized. There is also a clear gap in the quality of the design infrastructure, e.g. CAD tools, libraries, etc. In addition, the level of synchronous design experience dwarfs the small experience base in asynchronous circuit design. The individual chapters in this book are indications that this gap is narrowing. The asynchronous circuit discipline is becoming more viable even though much work remains to be done before we can hope to compete in the commercial sector with synchronous design styles.

# 2 Signaling Protocols

Most asynchronous circuit signaling schemes are based on some sort of protocol involving *requests*, which are used to initiate action, and *acknowledgments*, which are used to signal completion of an action. These control signals provide all of the necessary timing information to properly sequence the computational events in the system. The resulting computational model is very much like the dataflow model [36, 1] where the arrival of the necessary operand data triggers the operation. Similarly there is a concept of a sender of information and a corresponding receiver. From the circuit perspective, and ignoring data transmission issues for now, these request and acknowledge control signals typically pass between two modules of an asynchronous system. For example let there be two modules, a sender **A** and a receiver **B**. A request is sent from A to B to indicate that A is requesting some action by B. When B is done with either the action or has stored the request, it acknowledges the request by asserting the acknowledge signal which is sent from B to A. Most asynchronous signaling protocols require a strict alternation of request and acknowledge events.

There are several choices of how these alternating events are encoded onto specific control wires. Two choices have been so pervasive that they will be described here to illustrate the concept. One common choice is the *4-cycle* protocol shown in Figure 1. *RZ* (return to zero), *4-phase*, and *level-signaling*

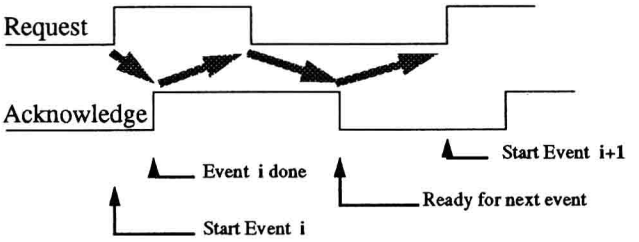


Figure 1: 4-cycle Asynchronous Signaling Protocol

are other names that have been used to identify this protocol. In Figure 1, the waveforms appear periodic for convenience but they do not need to be so in practice. The bold gray arrows indicate the required sequence of events. There is no implicit assumption about the delay between successive events. Note that in this protocol there are typically 4 transitions (2 on the request and 2 on the acknowledge) required to complete a particular event transaction. Proponents of this scheme argue that typically 4-cycle circuits are smaller than they are for 2-cycle signaling, and that the time required for the falling transitions on the request on acknowledge lines do not usually cause a performance degradation since they happen in parallel with other circuit operations, or are useful for controlling the transmission of the answer back to the requester.

The other common choice is *2-cycle* signaling shown in Figure 2, also called *transition* or *NRZ* (non-return to zero) signaling. In this case the waveforms are the same as for 4-cycle signaling with the exception that every transition on the request wire, both falling and rising, indicates a new request. The same is true for transitions on the acknowledge wire. 2-cycle proponents argue that that 2-cycle signaling is better from both a power and a performance standpoint, since every transition represents a meaningful event. The disadvantage is that most 2-cycle interface implementations require more logic than their 4-cycle equivalents. 2-cycle signaling is particularly useful for high-speed micropipelines as pointed out by Ivan Sutherland in his Turing Award paper [124].

So far, the discussion has only addressed control signals. There are also choices for how to encode data. A common choice is the use of a *bundled*

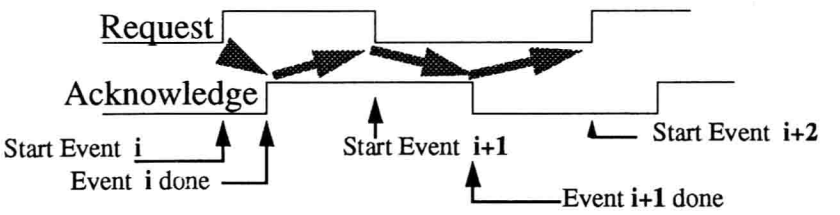


Figure 2: 2-cycle Asynchronous Signaling Protocol

protocol which can be either 2- or 4-cycle. In this case if an n-bit data value is to be passed from the sender to the receiver then n+2 wires will be required (n bits of data, 1 request bit, and 1 acknowledge bit). While this choice is conservative in terms of wires, it does contain an implied timing assumption. Namely the assumption is that the propagation times of the control and data lines are either equal or that the control propagates slower than the data signals. A sending module will assert the data wires and when they are valid will assert the request. It is important that the same relationship of data being valid prior to request assertion be observed at the receiving side. If this were not the case the receiver could initiate the requested action with incorrect data values. This requirement is often simply called the *bundling constraint.* Most asynchronous circuits have been designed with bundled data protocols because the logic and wires required to implement bundled data circuits is significantly less than with non-bundled approaches. However, in order for bundled data asynchronous circuits to work properly, the bundling constraint must be met. Antagonists of this approach note that these timing assumptions are similar to those made for synchronous circuit design.

The common alternative to the bundled data approach is *dual rail* encoding. In this case each bit of data is encoded with its own request onto 2 wires. A typical 4-cycle dual rail encoding has four states:

1. 00 - Idle, data is not valid
2. 10 - Valid 0
3. 01 - Valid 1
4. 11 - Illegal

After a valid data value is asserted the wires must return to the idle state prior to assertion of the next valid datum. In this case for an n-bit data value the link between sender and receiver must contain 2n+1 wires: 2n for the n-bits of data and the associated requests and 1 for the acknowledge. In this dual rail protocol, sending a bit requires the transition from the Idle state to either the valid 0 or valid 1 state and then after receiving the acknowledge it must transition back to the idle state. The illegal state is not used. If recognized by the receiver, it should cause an error.

A 2-cycle variant would still require 2 wires per bit but could signal a valid 0 by a single transition of the left bit, while a valid 1 would be signaled by a transition on the right bit. Concurrent transitions on both the left and right bits are illegal. Sending a 0 or a 1 must be followed by a transition on the acknowledge wire before another bit can be transmitted. Alternative encoding schemes have been proposed as well [139, 41]. Dual rail signaling is insensitive to the delays on any wire and therefore is more robust when assumptions like the bundling constraint cannot be guaranteed. The receiver will need to check for validity of all n-bits before using the data or asserting the acknowledge.

# 3 Completion Signals

One of the added complexities of asynchronous circuits is the need to generate completion signals that often correspond to the acknowledge signal in the various signaling protocols. There are many methods, none of which is universally satisfactory. A common approach is to design an asynchronous module in a manner that is similar to a synchronous circuit. Namely the arrival of the request starts the modules internal clock generator and after a certain number of internal clocks the circuit is done and an acknowledge can be generated. The idea was originally suggested by Chuck Seitz and was used during the construction of the DDM1 dataflow computer [36]. This technique works well when the size of the module is large, but when the module is small, the additional logic required for the internal clock generator represents an overhead that is too costly.

Another choice for completion signal generation is the use of a *model delay.* In this case, conventional synchronous timing analysis of the datapath is used to determine how long the circuit will take to compute a valid result after the request has been received. A delay element such as an inverter chain is then used to turn the request into the appropriately delayed acknowledge signal. Note that this method works equally well for both 2- and 4-cycle signaling protocols.

Special functions often have unique opportunities. For example, arithmetic circuits can be built to generate completion signals based on carry propagation patterns [62]. Other functions can independently compute both $F$ and $\overline{F}$ and use the exclusive-OR of their outputs to generate the acknowledge signal. Note that this technique will only work directly in a 4-cycle signaling protocol. If used for a 2-cycle protocol, then additional logic such as a T flip-flop will be required.

Another common case is when multiple subsystems are activated in parallel by a master controller. When all of their completion signals are asserted then the master controller should be acknowledged. A tree of C-elements is commonly used to provide the appropriate control in this situation.

A novel technique was proposed by Mark Dean [40] where completion detection was performed by observing the power consumption of the circuit. When activated the circuit consumes power and when it is done the power consumption falls below a particular threshold.

The study of completion signal generation methods in asynchronous circuits could be the topic of an entire book. For now, it is only necessary to realize that some method must be chosen and that the need for completion signals and related signaling protocols is a necessary overhead of asynchronous circuit design.