Reliable Interface Design for Combining Asynchronous and Synchronous Circuits

Lüli Josephson Erik L. Brunvand Ganesh Gopalakrishan John F. Hurdle
Computer Science Department
University of Utah
Salt Lake City, Utah 84112
E-mail: {luli,elb,ganesh,hurdle}@cs.utah.edu

Abstract - **In order to successfully integrate asynchronous and synchronous designs, great care must be taken at the interface between the two types of systems. Synchronizing asynchronous inputs with a free running clock can cause well-known problems with metastability in the synchronization circuits. Stretchable clocks allow a clock cycle to expand dynamically in response to the metastability effects of sampling asynchronous inputs. We use an interface organization where the special circuitry for detecting metastability and for stretching the clock that is delivered to the synchronous part of the system is encapsulated in a Q-flop-based interface. This provides a very convenient method for interfacing mixed systems, as the interface and clock generation circuitry are isolated into one special module, and neither the asynchronous nor the synchronous system need be modified internally to accommodate the interface. This is especially important when standard synchronous components are used as there is no opportunity to modify these parts. We show that this interface module is suitable for most mixed design needs and conclude with an example.**

1 Introduction

As VLSI technology improves, hardware systems become larger, faster, and more complex. Along with these improvements, however, come many problems directly associated with the speed and scale of the new systems. Asynchronous and self-timed design techniques are currently attracting renewed interest as a method for coping with some of the problems associated with the scale of modern systems. These systems that do not rely on a global clock to keep system components synchronized have been shown to exhibit a number of inherent advantages: more robust behavior (in terms of process and environmental variations), a capability for higher performance operation, decreased power consumption, and inherently higher reliability in high speed applications. However, for many applications, designing completely asynchronous circuits is currently an impractical approach. Because of the large body of work related to synchronous system design, and the many tools available for this style of design, mixed asynchronous-synchronous design is an important approach.

One domain where this mixed design style may be especially appropriate is in embedded systems and controllers. These systems typically use a commodity-type synchronous controller, but respond to a variety of external inputs that are not synchronized to the system clock. Processing the asynchronous input data as it arrives may be best accomplished using a specialized asynchronous circuit with overall system response being coordinated by the synchronous controller. When these devices are used in remote locations there is a real need for low power control circuits that are extremely robust under significant temperature and power supply variation. These types of systems present an inherent opportunity for highly parallel control interaction that must operate correctly and whose behavioral parameters are well proven and understood, due to the extreme remote nature of their operation and the resultant expense of repair.

In order to successfully integrate asynchronous and synchronous designs, great care must be taken at the interface between the two types of systems. Synchronizing asynchronous inputs with a free running clock can cause well-known problems with metastability in the synchronization circuits. To achieve this interfacing successfully, we implement an interface/clock generation module based on Q-flops, special flip-flops that signal when they have exited metastability, and stretchable clocks. Stretchable clocks allow a clock cycle to expand dynamically in response to the metastability effects of sampling asynchronous inputs. In our scheme the special circuitry for detecting these events and for stretching the clock that is delivered to the synchronous part of the system is encapsulated in a Q-flop-based interface, as an extension of Rosenberger's Q-Module organization [24]. This provides a very convenient method for interfacing mixed systems as the interface and clock generation is isolated into one special module, and neither the asynchronous nor the synchronous system need be modified internally to accommodate the interface. This is especially important when standard synchronous components are used as there is no opportunity to modify these parts.

In spite of the natural inertia from the mass of existing synchronous designs, there is currently a renaissance in asynchronous circuit design. As one aspect of our asynchronous systems design work, we look for ways to exploit the rediscovered advantages of asynchronous designs while capitalizing on the preponderance of previously designed synchronous parts. One approach involves designing globally asynchronous, locally synchronous systems (hereafter called "mixed systems"). We use Q-module-based interface/clock generation elements as shown in Figure 1 and described in Section 3 to safely coordinate and synchronize interactions between the different regions in these mixed systems. Unlike other value-reliable [9] synchronization circuits such as stretchable or stoppable clock methods, Q-modules (internally clocked, delay insensitive circuits) conform naturally to the self-timed asynchronous design style that we advocate in our asynchronous systems research.

One goal for this work is to devise an interface scheme that will allow us to use previously designed synchronous parts "as is". (We consider this category of synchronous parts to include off-the-shelf parts, design library modules, and modules synthesized from a behavioral specification by a high-level synthesis system.) This allows us to focus our efforts on the (to us) more interesting asynchronous design aspects yet avail ourselves as necessary of existing synchronous circuits.

The asynchronous circuits we consider in this paper are all built using two-phase delay-insensitive (DI) control circuits (the individual control modules may be speed independent, but exhibit delay-insensitive interfaces) and bundled data paths.

This paper describes our approach to building globally asynchronous, self-timed systems that incorporate locally synchronous regions. First we discuss mixed-system design issues, then review typical synchronization approaches and list their limitations. Next we present a Q-module-based interface implementation which combines synchronization and clock gen-

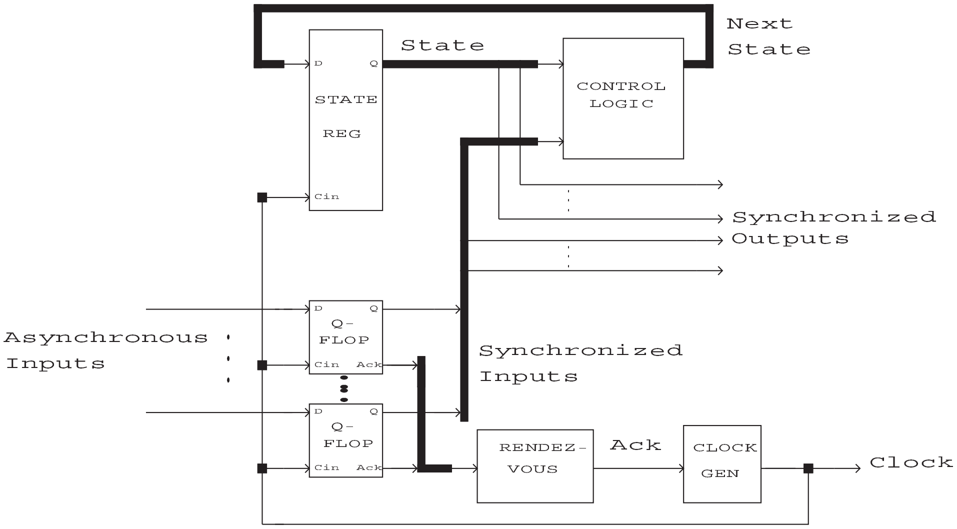

Figure 1: Q-module organization.

eration and facilitates the mixed-system design style we are advocating. We follow this by an example using the interface in a mixed asynchronous-synchronous implementation for a Finite Input Response (FIR) filter. Finally, we draw some conclusions about the method and indicate future extensions to this work.

2 Mixed Systems - Combining Asynchronous and Synchronous Regions

As system sizes and circuit technology speeds increase, it becomes more difficult to satisfy the high frequency clock requirements of robust, hazard- and race-free clock schemes and small skew global clock distribution [3]. Using autonomous locally clocked and/or asynchronous subsystems can facilitate large system design. As a corollary, global communication requirements are reduced since most communications can be localized. However, communications among synchronous systems not sharing a common clock must be synchronized.

2.1 Synchronization Issues in Mixed Systems

In synchronous systems, whenever flip-flop timing constraints such as minimum pulse width, setup, or hold times are not met, there is a finite probability that instead of resolving to a state matching the input value, the flip-flop will enter a metastable state. This metastable state may eventually resolve to the correct output state, or it may resolve to an incorrect output state. If it takes longer than one clock period for the metastability to resolve, and the synchronous system is allowed to proceed, the metastable output may be interpreted as a “1” by part of the system and “0” by the rest. We refer to this effect as “value failure”.

Metastability occurrence is probabalistic and exponentially related to the synchronous clock rate and the asynchronous data rate: the mean time between failure is shorter at high asynchronous data rates and high fast synchronizing clock frequencies. Conversely, the longer one waits before using the sampled data, the lower the probability of synchronizer failure. Conventional techniques for decreasing the probability of metastable failure [17, 20] include using fast devices, allowing for extended settling times, and employing masking and redundancy. These techniques all amount to allowing a long but bounded amount of time for metastability to resolve, accepting the finite probability that failures will occur. Thus these methods trade the advantage of time-certainty for the possibility of value failure.

2.2 Conventional Interface Techniques

Because of its probabilistic nature, metastability is hard to characterize, detect, and reproduce. As circuit technologies improve and circuit speeds increase, handling metastability issues becomes more important. Although the consequences of ignoring its effects are potentially catastrophic, it is still considered a very difficult problem to treat correctly in the design of interface circuits. Evidence for this is the number of interface specification systems that completely disregard this issue. For example, a range of interface specification methods are being developed in conjunction with several high-level synthesis systems. The implementations synthesized from these descriptions either include conventional synchronizers on the asynchronous inputs [2, 4, 13], or only handle signals related to the same clock internally or externally [12, 18, 21]. For high-speed, value-critical system interfaces, these methods cannot guarantee the required combination of value-reliability plus speed.

2.3 Reliable Value Synchronization Methods

Alternatively, less conventional methods using stoppable or stretchable clocks trade a degree of time-uncertainty for value-reliability. *Stoppable* clocks [9, 22, 25, 27, 28, 31] can be used for interfacing asynchronous-synchronous circuits communicating via specific protocols and when the synchronous system does not require a continuously running clock. The locally generated clock is stopped synchronously, and the system passively awaits the next asynchronous input. The clock is then re-started asynchronously by the asynchronous input with *known* value but *unknown* arrival time. For circuits that operate using this restricted interface model, this scheme avoids metastability because it avoids sampling a changing signal. However, stoppable clock circuits require careful delay analysis in their design to ensure correct circuit operation [9, 31]. Often redesign of the synchronous part itself is required as well. Thus stoppable-clock based interface schemes are applicable only in certain restricted cases.

Stretchable clock circuits can be used in the more general case when neither the value nor the arrival time of the incoming asynchronous signal is known in advance. In stretchable clock circuits [9, 10, 22, 28], a comparator-based metastability detector is used to delay the next clock edge when a potentially changing input is sampled. Again, critical timing conditions must be met in the design and operation [24].

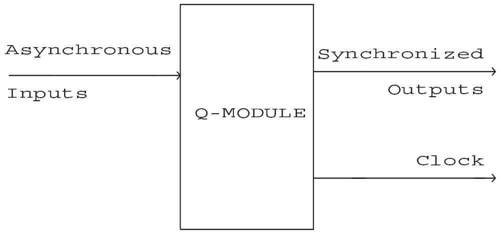

Figure 2: Q-Module-based interface.

3 Q-Module-Based Interfaces

The goal is to build value-safe interfaces for combining synchronous and asynchronous designs. The main difficulty with the stoppable/stretchable clock based approaches is that they require specialized and somewhat hard-to-build circuits. Yet interfaces based on these methods would facilitate the decoupled mixed-design style we are advocating, allowing us to incorporate synchronous parts into an asynchronous design without redesign of the synchronous parts. Because they are the more general of the two schemes, we focus primarily on stretchable clock techniques for our interfaces.

We implement a stretchable clock interface where the metastabillty detection circuitry is encapsulated in a special module called a Q-module, shown in Figure 1 and described in Rosenberger, et al., [24]. Q-modules are internally clocked, delay-insensitive state machines. Data inputs are sampled on one clock event using Q-flops: these special flip-flops use comparator-based circuits to acknowledge when metastability has resolved and values are stable. The acknowledgements from the Q-flops and the state registers are used to generate the next clock event.

We extend the Q-module organization by altering the Clock Generation element to provide the appropriate clock signal to the synchronous side of the interface. Note that in the normal case the period of this exported clock signal is regular and set by the delay through the clock generator element (with any additional delay needed to model the combinational delay time, or to achieve a particular minimum clock frequency). If the internal Q-flops stall, waiting to resolve metastability due to asynchronous inputs, the clock signal will be stretched automatically until the data resolve.

In contrast to conventional stretchable clock implementations, because Q-flops acknowledge when they are ready to accept new inputs as well as acknowledging when the storage operation is completed, the clock generation utilizes a delay-insensitive protocol. Clock generation and distribution are also simpler: delay through the combinational logic implementing the state function provides the single delay constraint that must be satisfied in the clock generation and distribution for the Q-module itself. For our interfaces, we use Q-modules both to synchronize the asynchronous inputs and to act as clock generators. The internal two-phase single wire clock, used for internal sequencing, is exported to form the synchronous part clock input, as is shown in Figure 2. Thus we have to adjust the single delay constraint in the Clock Generation element to provide the minimum clock frequency required by the synchronous part.

The goal for this work is to demonstrate an interfacing method to allow value-reliable synchronization between self-timed asynchronous and conventional synchronous systems. In contrast to the stretchable and stoppable clock methods discussed above, Q-modules provide the safety, generality, and simplicity required of an easily designed interface. Our Q-module interface/clock generation circuits avoid synchronization failure by ensuring that storage elements have resolved before using the stored values by dynamically adjusting the additional delay.

Contrast this with the fixed time delay that is normally introduced for synchronization in conventional clocked circuits (for example, by including extra latching stages or dividing down the clock for sampling latches). In the Q-module-based interface, value-safety is assured at the expense of an occasional stretching in the clock period. In conventional clocked circuits, the clock period must always be long enough to allow sufficient time for metastability to resolve with as high a probability as possible. Yet because the resolution time is fixed and bounded in conventional techniques, there is always a finite probability of value failure and subsequent circuit failure.

4 An Example Mixed-System Design

Now we discuss a relatively small example - a mixed asynchronous-synchronous circuit implementation for an FIR filter. This circuit incorporates a Q-module-based interface for value-safe synchronization and clock generation. This circuit contains all the typical elements of a larger mixed-system design, but is simple enough to allow us to focus on the important interface details. For example, synchronous arithmetic parts are inexpensive, generally compact, and readily available. Their use can be very cost effective in a circuit which is otherwise asynchronous. The FIR filter circuit below is designed to operate in an asynchronous environment and so it must present an asynchronous interface to the external world. However, in addition to the asynchronous data buffering parts, it uses synchronous computation parts internally and so requires synchronization at that internal interface. The asynchronous control structure will allow the data-dependent processing time of the required arithmetic calculations to enhance the average-case performance of the filter.

The circuit described represents a class of circuits where mixed design is appropriate. For example, embedded control using standard synchronous controllers responding to asynchronous events are an excellent target for the synchronization strategy we describe here.

4.1 Circuit Description

We implement a universal FIR filter for asynchronous inputs. This is a sixth order filter, with input and output done in parallel. The symmetric coefficient algorithm is used which reduces the number of multiplications by a half over the standard filter. For the 4-bit input vector x_n and for three 4-bit coefficient vectors k_a, k_b, k_c, the 11-bit output vector y_n is computed as:

$$y(n) = k_a * [x_n + x_{n-5}] + k_b * [x_{n-1} + x_{n-4}] + k_c * [x_{n-2} + x_{n-3}]$$

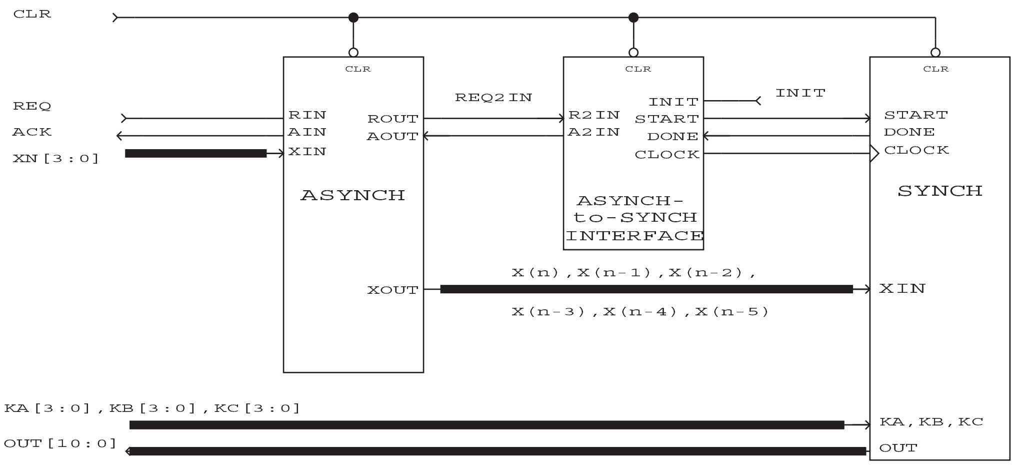

Figure 3: Top-level diagram for mixed asynchronous-synchronous FIR implementation.

The FIR filter circuit as shown in Figure 3 consists of an asynchronous data-fetching unit Asynch and a synchronous computation unit Synch. A Q-module-based interface part Asynch-to-Synch Interface between the asynchronous module and the synchronous module links the control of both parts, synchronizes the data, and generates the clock for the synchronous part. The external interface is self-timed using two-phase transition signaling and data-bundling. The environment issues a Req when the next input data value is ready. The FIR returns an Ack when the FIR computation completes and a new output value is available. The asynchronous self-timed interface constrains event sequencing rather than event timing. The environment can supply data and use results at any rate as long as the sequence of "send-Req; receive-Ack" is obeyed.

The data-fetching part Asynch is an asynchronous shift register with 6 data taps implemented as a Sutherland micropipelined-based FIFO [29]. It uses two-phase self-timed control with data-bundling.

The Q-module-based interface part Asynch-to-Synch Interface, Figure 4, provides synchronized asynchronous data and the clock for use by the synchronous computation unit. A Q-flop is used to detect metastability when latching the "begin next computation" control signal REQ2IN is received from the asynchronous part. The clock generation circuit waits for metastability to resolve before initiating the next clock pulse and latching the outputs. The clock is exported to the synchronous part and is continuously running. While the maximum period may vary, the minimum clock period to match the synchronous part delay is guaranteed by the delay in the clock generation circuit.

Since the asynchronous interface communicates via a two-phase protocol, and the synchronous interface obeys a four-phase protocol, the interface circuit includes a 2-to-4 phase protocol converter part. This asynchronous converter circuit takes the place of the combinational logic implementing the state function in a regular Q-module. For this very simple interface, no additional logic or state-holding Q-flops are required. This is not usually the case. We are working on more complicated examples where state information and feedback within the Q-module are required [11, 16].

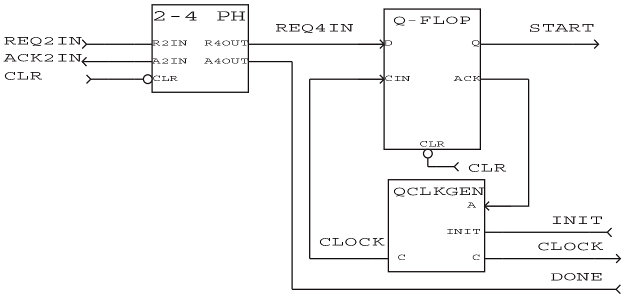

Figure 4: Asynch-to-Synch Interface Implementation

The synchronous computation unit SYNCH consists of five adders and three synchronous multipliers. The multipliers have data-dependent completion times, so the number of cycles for each FIR computation will vary. Output data are latched, and so remain valid until the next FIR computation starts. For historical reasons, this module obeys a four-phase synchronous self-timed protocol.

4.2 Implementation Details

A prototype version of the mixed asynchronous-synchronous FIR filter has been designed for implementation in Actel FPGAs [1]. It was built and functionally simulated using the asynchronous macro cells described in [6] and the synchronous parts provided as Actel Macro Library soft macros, and contains 640 logic modules.

Q-flops used in the interface are not available as standard digital parts. Although Q-flops cannot be built using FPGAs because it is not possible to implement the required analog comparator circuit, we have designed circuits that mimic the behavior of a Q-flop using Actel FPGAs [6]. These FPGA circuits use a fixed delay to resolve metastability [1] and so are not value-safe, but they allow us to prototype designs quickly that can then be upgraded to use custom Q-flops as required.

We have built and tested actual Q-flops in a variety of technologies including CMOS [5, 11], and Gallium Arsenide [7]. The next step is to implement the asynchronous parts of this interface design in another technology where the actual analog metastability resolver parts can be used.

5 Results and Future Work

Q-flops are found in several circuit libraries [5, 7, 8, 19, 24, 26]. Combining Q-flops with control logic to form delay-insensitive state machine modules (Q-modules) has been extensively described by Rosenberger, et al., in [24]. Sutherland and Sproull suggest such an asynchronous state machine built using Q-flops as an asynchronous replacement for a conventional register-based synchronizer plus synchronous state machine in a three port memory controller in [30]. In this paper, we introduce a Q-module-based interface circuit for value-safe synchronization and clock generation. We demonstrate its use in a mixed asynchronous-synchronous FIR circuit implementation.

In general, these interface modules can be used in mixed systems whenever synchronization and clock generation are required and the synchronous element can tolerate some occasional clock elongation. Value-safety is assured. Chapiro [9] has done studies that indicate clock speeds can actually be increased, now that synchronization delay can be "dynamically adapting". In contrast to other recent work [2], no redesign of the synchronous part is required to accommodate it in the globally asynchronous self-timed environment. This makes the method especially suitable for correctly including previously designed synchronous parts in mixed systems.

There are many kinds of mixed systems where these interfaces can be used. This paper describes one example circuit containing asynchronous and synchronous regions that need to synchronize. Traver [31] describes a method for interfacing multiple locally-synchronous but mutually-asynchronous circuits. Synchronization problems arise in embedded controllers using synchronous circuits but responding to asynchronous events. These types of systems are often installed in remote locations and must be robust under a variety of environmental conditions that tend to make synchronization problems worse. Our technique could be used in both of these applications.

The FIR is a small mixed-system demonstration circuit. As asynchronous systems evolve in size, such an approach will become a necessity until suitable asynchronous computation components are commonly available. One system-size example we are working on in our lab [16] is interfacing a synchronous off-the-shelf math coprocessor chip to a fully self-timed asynchronous microprocessor [23]. Yet another example involves connecting synchronous analog-to-digital input converters and digital-to-analog output converters to an asynchronous neural net classifier system [14, 15] also being constructed in our lab. The issues involved in successfully combining more than two systems using these interfaces are being explored, as described in [11].

For a variety of reasons, including limitations in current asynchronous design tools, designing completely asynchronous circuits may not always be practical or desirable. With the addition of the interface method described here, the self-timed circuit design style we advocate allows us to design and implement globally asynchronous designs that incorporate correctly synchronized locally synchronous regions. Mixed systems retain many of the high-level advantages of asynchronous designs, while exploiting existing synchronous designs.

References

[1] Actel Corporation. *ACT Family Field Programmable Gate Array Databook*, March 1991.

[2] Naser Awad and David Smith. Automatic interfacing of synchronous modules to an asynchronous environment. In *International VLSI Conference*, 1991.

[3] H. B. Bakoglu. *Circuits, Interconnections, and Packaging for VLSI.* Addison-Wesley, 1990. Chapter 8.

[4] Gaetano Borriello. Specification and synthesis of interface logic. In R. Camposano and W. Wolf, editors, *High-Level VLSI Synthesis*, chapter 7, pages 153–177. 1991.

[5] Erik Brunvand. Parts-r-us. *a chip aparts(s)....* Technical Report CMU-CS-87-119, Carnegie Mellon University, May 1987.

[6] Erik Brunvand. A cell set for self-timed design using actel FPGAs. Technical Report UUCS–91–013, University of Utah, 1991.

[7] Erik Brunvand, Nick Michell, and Kent Smith. A comparison of self-timed design using FPGA, CMOS, and GaAs technologies. In *International Conference on Computer Design*, Cambridge, Mass., October 1992.

[8] Erik Brunvand and Robert F. Sproull. Translating concurrent programs into delay-insensitive circuits. In *ICCAD-89*, pages 262–265. IEEE, November 1989.

[9] Daniel M. Chapiro. *Globally-Asynchronous Locally-Synchronous Systems.* PhD thesis, Department of Computer Science, Stanford University, October 1984.

[10] Daniel M. Chapiro. Reliable high-speed arbitration and synchronization. *IEEE Transactions on Computers*, C-36(10):1251–1255, October 1987.

[11] Ganesh Gopalakrishnan and Lüli Josephson. Towards amalgamating the synchronous and asynchronous styles. In *Tau 93: 1993 Workshop on Timing Issues in Specification and Synthesis of Digital Systems*, September 1993.

[12] M.R. Greenstreet and K. Li. Simple hardware for fast interprocessor communication. Technical Report CS-Tr-242-90, Princeton University, January 1990.

[13] S. Hayati and A. Parker. Automatic production of controller specifications from control and timing behavioral descriptions. In *Proceedings of the 26th ACM/IEEE Design Automation Conference*, pages 75–80, 1989.

[14] John F. Hurdle. Self-timed neural model implementation: an example using CMAC. In *Proceedings of the 26th Hawaii International Conference on Systems Science*, pages 369–378, January 1993.

[15] John F. Hurdle, Lüli Josephson, Erik L. Brunvand, and Ganesh Gopalakrishnan. Asynchronous models for large scale neurocomputing applications. In Jean-Claude Rault, editor, *Neural Nimes 92: Proceedings of the Fifth International Conference on Neural Networks and their Applications*. 1992.

[16] Lüli Josephson. Mixed asynchronous and synchronous circuits. Master's thesis, Computer Science Department, University of Utah, 1993. In preparation.

[17] Lindsay Kleeman and Antonio Cantoni. Metastable behavior in digital systems. *IEEE Design and Test of Computers*, pages 4–19, December 1987.

[18] David Ku. *Constrained Synthesis and Optimization of Digital Integrated Circuits from Behavioral Specifications.* PhD thesis, Department of Computer Science, Stanford University, June 1991.

[19] C. E. Molnar, T. P. Feng, and F. U. Rosenberger. Synthesis of dely-insensitive modules. In *Proc. 1985 Chapel Hill Conference on Very Large Scale Integration.* Computer Science Press, March 1985.

[20] Bryon I. Moyer. Has the metastability world stabilized? In *Proceedings of the Second Annual PLD Design Conference and Exhibit*, 1992. April.

[21] J. A. Nestor and D. E. Thomas. Behavioral synthesis with interfaces. In *IEEE International Conference on Computer-Aided Design, ICCAD-86: Digest of Technical Papers*, pages 112–115, 1986.

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

[23] William F. Richardson and Erik Brunvand. The NSR processor prototype. UUCS Tech Report UUCS-92-029.

[24] Fred U. Rosenberger, Charles E. Molnar, Thomas J. Chaney, and Ting-Pein Fang. Q-modules: Internally clocked delay-insensitive modules. *IEEE Transactions on Computers*, 37(9):1005–1018, September 1988.

[25] Charles L. Seitz. System timing. In C. Mead and L. Conway, editors, *Introduction to VLSI Systems*, chapter 7. Addison Wesley, Reading, MA, 1980.

[26] Robert F. Sproull. Qflop modules. Technical Memo 4472, Sutherland, Sproull and Associates, June 10, 1986.

[27] Robert F. Sproull and Ivan E. Sutherland. Stoppable clock. Technical Memo 3438, Sutherland, Sproull and Associates, Jan. 9, 1986.

[28] M.J. Stucki and J.R. Cox, Jr. Synchronization strategies. In *Proceedings of the Caltech Conference on VLSI*, pages 375–393, January 1979.

[29] Ivan Sutherland. Micropipelines. *CACM*, 32(6), 1989.

[30] Ivan E. Sutherland and Robert F. Sproull. Three port memory. Technical Memo 4497, Sutherland, Sproull and Associates, Sept 1, 1986.

[31] Cherrice Ann Traver. A testable model for stoppable clock ASICs. In *Proceedings of the Fourth Annual IEEE International ASIC Conference and Exhibit*, pages P6–3.1, 1991.