# An On-Chip Dynamically Recalibrated Delay Line for Embedded Self-Timed Systems

George Taylor, Simon Moore, Steev Wilcox, Peter Robinson
*Computer Laboratory, University of Cambridge*
{*gst22,swm11,spw14,pr*}*@cl.cam.ac.uk*

## Abstract

*Self-timed systems often have to communicate with their environment through a clocked interface. For example, off-chip memory may require clocking and this can reduce the benefits of self-timed design. This paper presents the design of a delay line which may be used to control the timing of an off-chip interface. Timing accuracy is maintained by periodically recalibrating against a low frequency reference clock. The design uses two delay lines so that one can be recalibrated while the other is in use. Recalibration is undertaken once each second; power consumption is low as the calibration circuitry is dormant most of the time. A particular implementation of the design is presented which is suitable for a standard cell or FPGA technology, together with experimental performance figures. The paper concludes with some remarks about possible applications in low-power synchronous design.*

## 1. Introduction

Interfacing self-timed systems to clocked devices, in particular to off-chip memory, can be frustrating because many of the benefits of self-timing can easily be lost. For example, using a high frequency off-chip oscillator to time off-chip devices would increase power consumption and electromagnetic emissions. Such a solution also requires synchronisation with the self-timed part, and may require a warm-up period from standby (typically 1ms for a crystal oscillator). An off-chip reference delay device, which is calibrated at the time of manufacture, can be used. However, this consumes a great deal of power, typically about 35mA whilst powered up [1] and adds an extra component to the system. A simple fixed length on-chip delay line requires a large error margin to meet minimum timing requirements regardless of manufacturing tolerances and environmental conditions. This error margin can severely reduce system performance. The delay line could be calibrated post fabrication, for example with laser trimming, ROM, or pin configuration, but this adds expense.

An alternative approach is to calibrate an on-chip oscillator or delay line against an off-chip low frequency (and therefore low power) crystal oscillator. For example the Philips PCA-5010 micro-controller [2] uses on-chip 6MHz oscillator constructed from an inverter ring. A counter measures the number of oscillations within a 76.8kHz reference clock period. Application software can read this counter and adjust the oscillator frequency by controlling the supply current to the inverter ring. The oscillator is used to time off-chip accesses and to clock a DC-DC converter. Related work in the synchronous design field includes a scheme for providing a dynamically calibrated delay between a global clock and registers sampling the device inputs [3] to compensate for inter-chip signal skew.

Many embedded systems use a low frequency (32kHz) crystal oscillator to maintain a real-time clock. We propose to use this existing time reference to dynamically calibrate an on-chip delay line, with the intent of making calibration as transparent to the user of the delay line as possible, within a standard cell implementation. In normal operation an adjustable delay line is used to time each off-chip memory access. During calibration the delay line is used as an oscillator and the number of oscillations within one period of the 32kHz reference clock is counted. Amulet3i uses a similar technique [4] but requires software level calibration during which off-chip accesses must be avoided. The control structure for the Amulet3i delay line may require less chip-area, provided that asymmetric 3-input C-elements are available.

The next sections describe the circuitry involved. First a simple tunable delay line is introduced. A ‘fast-mode’ feature is then added to speed up calibration at initial reset. Two of these delay lines are then combined into a double-buffering scheme such that whilst one delay is being recalibrated the other is available for use. The calibration and control system is then discussed. The penultimate section presents experimental data obtained from a test implementation on an FPGA device before the final section gives some concluding remarks.

## 2. Tunable delay

The delay line shown in Figure 1 (input `din` output `dout`), consists of a sequence of delay cells, shown in Figure 2. Each delay cell consists of three sections: a delay element, some control circuitry and completion detection. The number of delay cells required depends on the implementation technology, the delay desired and the range of environmental conditions.

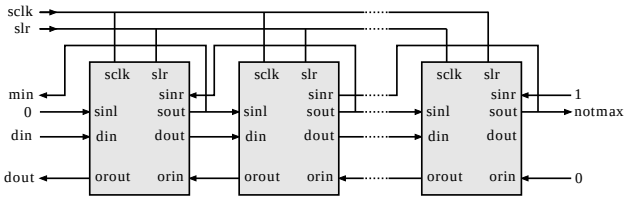


**Figure 1: Tunable delay line**

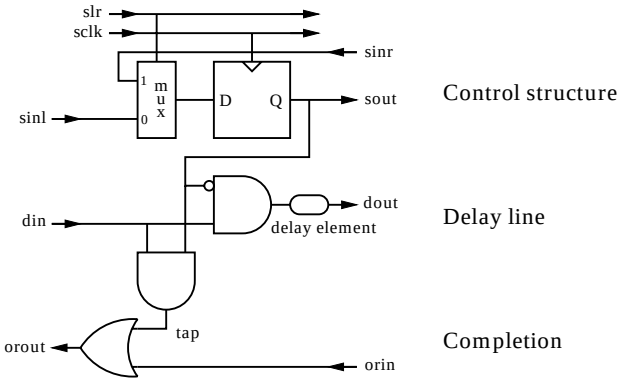


**Figure 2: Single delay cell**

Each delay cell can be in one of two modes depending on the flip-flop state `sout`.

**`sout` low** —The delay input `din` is delayed via the delay element and passed on `dout` to the next cell. This is repeated until a cell with `sout` high is encountered.

**`sout` high** —The delay input `din` is passed down to the completion stage on `tap`.

The delay element implementation is technology specific and depends on the desired granularity of the tunable delay. Using a slower delay element decreases the number of cells needed at the expense of large granularity and hence decreased accuracy. Note that only delay cells which are actually required are active, enabling many extra cells to be added to take care of extreme environmental conditions (such as a large supply voltage increase to cope with a demand for higher performance elsewhere in the system) without increasing power consumption during normal operation. The number of delay cells could be reduced by adding a constant, non-tunable, offset delay at the start of the delay line.

The completion section ORs together all the `tap` signals to produce the output from the delay line. A linear structure is used which delays the signal by an amount proportional to the tunable delay length. Other OR structures are possible but might incur large discontinuities in the tunable delay.

The control structure section forms a simple bidirectional shift register across the delay line. The shift direction is controlled by `slr` and the register is clocked from `sclk`. Zero's are shifted in from the left and one's are shifted in from the right. The state of the left and right most cells indicate if the tunable delay has reached the minimum or maximum delay possible. At initialisation all registers are cleared resulting in the maximum delay.

Initially we experimented with binary and Gray code counters instead of the shift register to reduce the number of state holding registers, however, this required extra decode logic and increased layout complexity. The shift register control signals, `slr` and `sclk`, can be timed with a generous margin without reducing availability, because double buffering is used (discussed in the next section).

During normal operation it should not be necessary to adjust the tunable delay often. For example, one shift left or right at rate of 1Hz should suffice. However, at initial switch-on, or after a sudden environmental change, it may be desirable to tune the delay quickly. A small amount of extra circuitry, shown in Figure 3, is added to detect when the last two shifts were in the same direction and the tunable delay has not reached the corresponding minimum or maximum (using the `min` and `notmax` signals). The `fastmode` signal then informs the control system that adjustment should be performed more frequently.

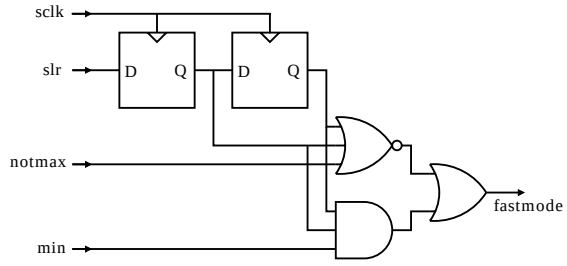


**Figure 3: Fastmode detection circuit**

## 3. Double buffering

Calibrating the tunable delay line requires two steps. Firstly the delay line is used as an oscillator and the number of oscillations within a reference period counted.

Secondly the delay line is adjusted. During calibration the delay line is unavailable for normal use. The control system described later takes several 32kHz clock periods to perform a calibration, which is an unacceptably long period of unavailability for our purposes. Therefore, we propose the use of two delay lines. Whilst one delay line is being calibrated the other is available for use. The delay lines are swapped over after each calibration. The delay line available for use will be at worst one second out of calibration.

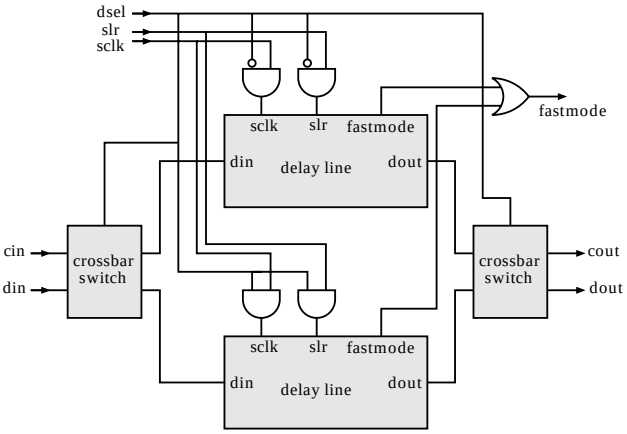


**Figure 4: Double buffered delay line**

Figure 4 shows a double buffered delay line. Signal `dsel` is used to select which delay is calibrated via `cin` and `cout` and which delay is available for use via `din` and `dout`. The shift register control signals are also demultiplexed according to `dsel`. Whilst it would be possible to indicate `fastmode` for each delay line separately, this would add unnecessary complexity. If one delay line requires faster calibration it is likely the other will as well. The crossbar switch is the simple circuit shown in Figure 5.

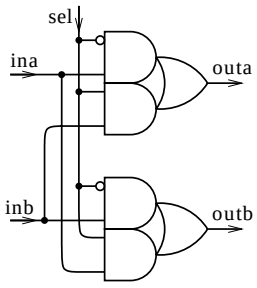


**Figure 5: Crossbar switch circuit**

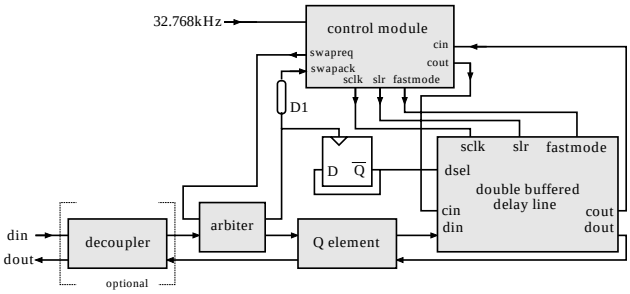


**Figure 6: Overall circuit**

## 4. Overall view

The overall system is shown in Figure 6. The toggle flip-flop in the middle holds the state of which delay is currently in use. Signal `swapreq` from the control module requests that the delays be swapped, `swapack` acknowledges, with a four phase-handshake. Delay `D1` matches the time required for the swap.

A period of unavailability, determined by `D1`, is necessary whilst the delays are being swapped. During such unavailability, a delay request on `din` may still be made but the request may be delayed by an extra amount. `D1` may be very conservative, to allow for the expected varying environmental conditions. This would not affect performance except during the infrequent swapping of delays. In practice `D1` need not be long and, when the delays present in the control module and arbiter are taken into account, may be omitted. The critical part of swapping requires changing of the cross-bar switches which involves a logic depth of one gate.

An arbiter is needed to arbitrate between use of the delay line (via `din`) and swapping the delay lines over. Since delays are swapped infrequently it is unlikely that arbitration is often required. It is the arbitration which may add extra delay to a request during swapping.

The Q-element [5] is used to send both rising and falling events through the delay line before acknowledging `dout`. The Q-element ensures that the arbiter is not released until the delay line has been through both rising and falling edge phases. Additionally, this permits the delay length to be doubled without requiring extra delay cells, but at the expense of reduced granularity. The STG and the circuit derived using Petrify [6] for the Q-element are shown in Figures 7 and 8.

The control module, described later, does not wait for `swapack`. Therefore, it is assumed that the arbiter will have successfully arbitrated and that the delay lines will have swapped over before the next calibration. Calibrations occur most frequently when `fastmode` is active, which is every $5 * 1/32.768kHz = 152\mu s$. This assumption is

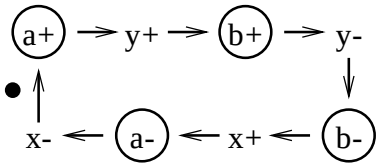


**Figure 7: Q-element STG**

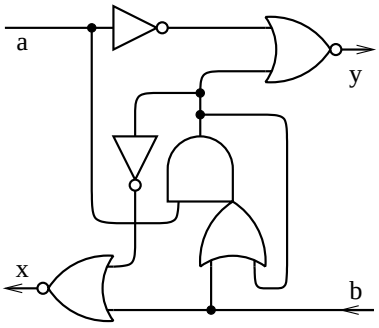


**Figure 8: Q-element circuit**

reasonable, 152$\mu$s is equivalent to over 300,000 gate delays in a typical $0.35\mu m$ process.

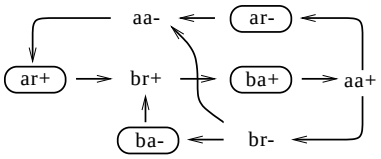


**Figure 9: Decoupler STG**

However, it is also possible that the environment chooses to keep `din` high for a long time. The swap would be delayed and it is then possible, that after some time, the control logic may attempt another calibration, during which the environment lowers `din`, resulting in the delays being swapped during calibration. The environment is hence required to lower `din` in good time to permit a swap to occur before the next calibration. Given that calibrations occur at most every 152$\mu$s, and that a typical delay is of the order of 10 to 100nS, such an assumption would seem reasonable. If not, the falling handshake decoupler circuit, shown in Figures 9 and 10, can be inserted to move the assumption from the environment to within our circuit, where it is easily met.

## 5. Control module

The control module referred to in Figure 6 is shown in Figure 11. The overall function is to time how many pulses can be passed through one delay line during one 32kHz reference clock period. Timing is normally performed once every second, unless fast-mode is active, in which

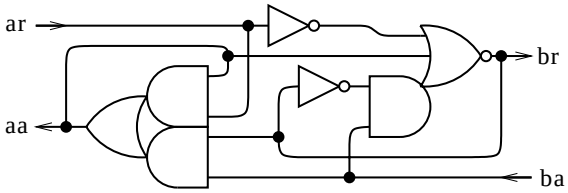


**Figure 10: Decoupler circuit**

case timing and adjustment are performed repeatedly until calibration is complete.

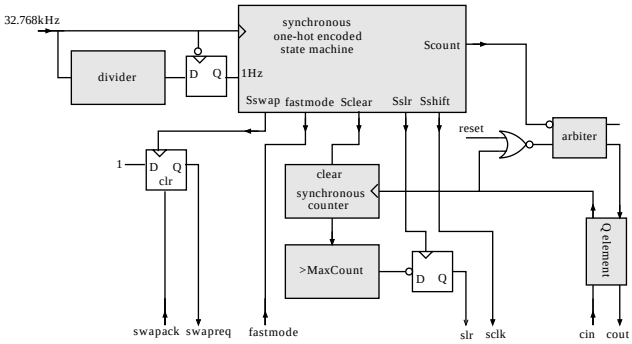


**Figure 11: Control module**

A synchronous state-machine, shown in Figure 12, clocked from the continuously running 32kHz reference clock forms the basis of the control module. The state machine is one-hot encoded to ensure that the state outputs, some of which are used to clock other flip-flops, are glitch free. A divider, consisting of a chain of toggle flip-flops, informs the state-machine when a calibration is due. The 1Hz signal is synchronised with the 32kHz clock falling to ensure proper timing when sampled. A ripple counter is used to count how many pulses can be transmitted through the delay line in a reference time period.

Operation is as follows. Initially the state-machine is in state `SwaitHz` waiting for the `1Hz` signal, upon which the next state `Sclear` clears the counter. The next state `Scount` enables the calibration loop, via the arbiter discussed below. At this point the loop formed around `cout`, `cin` and the delay line under calibration form an oscillator. The counter increments every oscillation, until the next control state `Swait` reasserts the upper request input to the arbiter stopping oscillation. The `Swait` state provides time for the counter and >MaxCount comparator to stabilise before the `Slr` state samples the comparison result. The result determines if the delay line shift register should move left to decrease the delay, or right to increase the delay. The next state `Sshift` clocks the shift register. State `Sswap` then triggers the `swapreq`, `swapack` handshake, controlled by a flip-flop. Finally another calibration is then performed if `fastmode` was set in

response to sclk, or if not the state machine waits for the next 1Hz signal.

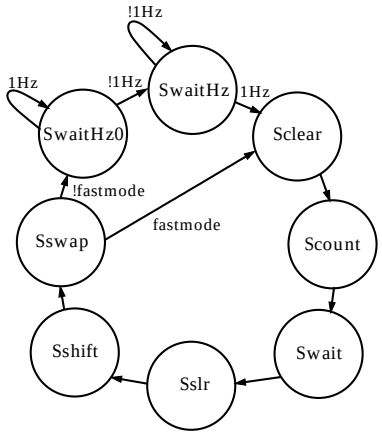


**Figure 12: One-hot encoded state machine**

There are several timing assumptions involved in this scheme. The shift register requires a setup time before clocking. The fast mode signal is likewise assumed to be valid before Sswap finishes. These and other typical assumptions required in clocked systems are easily met because the clock period of $30.5\mu$s is so long.

However, there is one other assumption worth noting, and an oddity surrounding the use of the arbiter which needs further explanation. If the arbiter (and input inverter) were replaced with a simple AND gate, it would be possible for the counter to receive a runt clock pulse, perhaps resulting in metastability on the comparator output. Although one $30.5\mu$s clock period would be available for the metastability to resolve before the comparator output would be sampled.

Instead an arbiter was added to ensure that the counter always receives a proper clock. The oscillator is stopped by requesting the upper arbiter input. However, there is no safe way for the clocked state-machine to wait for the corresponding grant output from the arbiter; the output is not connected. A problem would occur if the arbiter took just the right amount of time to arbitrate such that the counter was clocked just before the comparator output was sampled. For this to occur the arbiter would have to take an exceedingly long time to arbitrate ($30.5\mu$s). Therefore, as before it is assumed that the arbiter completes within a $30.5\mu$s period (equivalent to at least 100,000 gate delays). Note that a NOR gate with reset is required to ensure correct initialisation. Until now, the global reset signal has not been shown.

An alternative strategy would be to use a self-timed control circuit instead of the clocked state-machine. However, the synchronisation problem would move to the 1Hz input and would require additional circuitry to delay match or completion detect various activities such as the shift register setup time and the counter clear time. An asynchronous counter could then be employed but it was decided such an approach would be more complicated than necessary.

There is one further important reason for using the arbiter, and the Q-element. The delay path from din to dout in Figure 6 also involves these components. The only difference between the calibration delay path and the externally available delay path is the decoupler element in the real path and the NOR gate in the calibration path; these have fairly similar delays. The Q-element could be omitted from the calibration path, for example, if its presence means the counter clock is not low for long enough. However, having both the Q-element and arbiter in the calibration path increases the accuracy with which the calibration and real delay paths match.

## 6. Implementation

A test implementation was performed with a Xilinx XC4000 FPGA device. The delay element present in the delay cell (Figure 2) is implemented by constraining the AND gate preceding it to be in a unique CLB. The delay is thus formed from a CLB and routing between the previous and next cells in the delay line. The cell delay is approximately 7-8ns at ambient temperature in the device used. A delay length of 25 cells was chosen with a target delay of 120ns, which was found to require about 13 delay cells. Note that additional delay exists in the routing outside of the delay line. Also, clock buffers, not shown in diagrams, were inserted in appropriate places.

A component not available in the Xilinx architecture is the arbiter, which was implemented using the approach shown in Figure 13, a simplified version of that in [7]. A state machine performs decision making, clocked by a locally generated clock which is active whenever one of the ports is non-zero. It is assumed that the metastability on the input flip-flops will have resolved before the output flip-flops are next clocked. This is sufficient for a test implementation, but would obviously be replaced by a cleaner Seitz arbiter [8] for a CMOS implementation, or an FPGA with an Arbiter cell could be used [9].

Simple floor-planning, but not detailed layout, was performed, to try and ensure that circuitry specific to the calibration path has similar routing delays to circuitry specific to the real delay path. In the absence of full layout, routing differences are still likely to be a source of error.

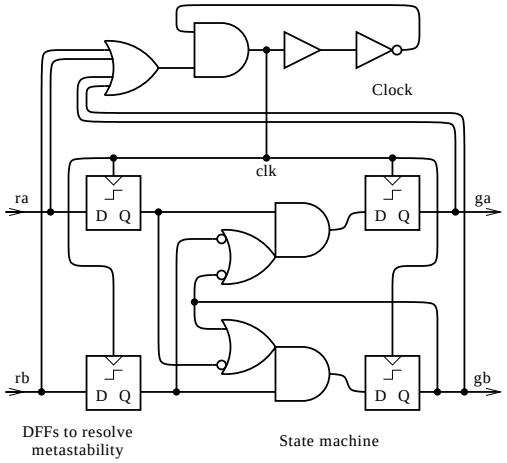


**Figure 13: Xilinx Arbiter Circuit**

## 7. Results

To test the implementation a pulse was sent through the delay line path (`din` to `dout`) after each delay swap. A record of the delay of each delay line (by extracting every second result) with varying environment was made. It is unwise to vary the supply voltage to a Xilinx device, as this might result in a corrupt configuration, I/O malfunction or other damage, so instead the temperature was varied using a hot air gun and freezer spray. The temperature range was approximately -35°C to 60°C.

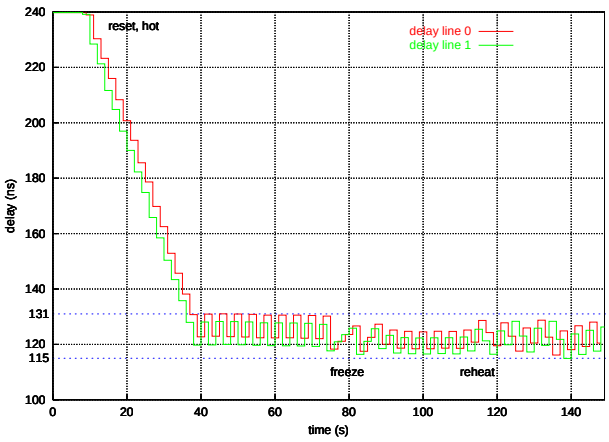


**Figure 14: Measured delay vs time for switch-on, freeze and reheat**

Figure 14 shows the results obtained. Two plots are shown, one for each delay line. To facilitate recording and viewing of the initial calibration the fast-mode feature was disabled. The graph divides into five chronological sections, initial switch-on (reset) whilst at a maximum temperature, continued heat, application of freezer spray, no heating or cooling and re-heating.

It can be seen that the delay lines are re-calibrated every second, switching between a delay which is slightly too large and slightly too small. Initially one delay line was switching between 10 and 11 delay elements and the other between 11 and 12, representing a difference in layout and routing between them. Another, expected, observation is that the step sizes are larger at a higher temperature because the delay of each cell increases with temperature. Around where the freezer spray is applied the delay increases in small steps and decreases in a large step. Each small increase represents one more delay element minus the effect of extra cooling, and a large decrease represents one less delay element plus extra cooling. If the fast-mode feature had been enabled, the first two successive steps up would have switched on fast-mode, speeding up compensation for the falling temperature.

The maximum and minimum delays obtained were 131ns and 115ns. From the graph it can also be deduced that a delay of 122-123ns was the calibration target. The 2-3ns discrepancy from the requested 120ns is due to a lack of careful layout and routing. The expected error is plus or minus one delay cell delay and the observed results confirm this. A somewhat lower error is thus expected from an ASIC implementation, where the granularity of the tunable delay is much smaller.

## 8. Conclusion

This paper has presented a dynamically calibrated delay-line suitable for low power self-timed embedded systems. The delay line uses a standard low frequency clock source as a reference. A double buffering scheme was used to increase availability. Experimental results performed using a FPGA implementation confirm the expected accuracy of the calibration across a widely varying temperature. An ASIC implementation would improve the accuracy due to both a finer granularity and a more balanced layout. Design extensions such as allowing the requested delay to be changed during operation are also possible. An alternative application of the delay line is in the provision of a high frequency clock for synchronous circuits, permitting an embedded micro-controller to resume instantly from a sleep state [10].

## References


[1] N. Components, *Digital delay modules 81A, 84A series Datasheet*.

[2] P. Semiconductors, *PCA5010 Data Sheet*.

[3] A. Marshall, B. Coates, and P. Siegel, "Designing an asynchronous communications chip," *IEEE Design & Test of Computers*, vol. 11, no. 2, pp. 8–21, 1994.

[4] J. G. et al, "AMULET3I an asynchronous system-on-chip," in *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, Apr. 2000.

[5] A. J. Martin, "Synthesis of asynchronous VLSI circuits," in *Formal Methods for VLSI Design* (J. Straunstrup, ed.), ch. 6, pp. 237–283, North-Holland, 1990.

[6] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Loavagno, and A. Yakovlev, "Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers," in *XI Conference on Design of Integrated Circuits and Systems*, Nov. 1996.

[7] S. W. Moore and P. Robinson, "Rapid prototyping of self-timed circuits," in *Proc. International Conf. Computer Design (ICCD)*, Oct. 1998.

[8] C. L. Seitz, "System timing," in *Introduction to VLSI Systems* (C. A. Mead and L. Conway, eds.), ch. 7, Addison-Wesley, 1980.

[9] S. Hauck, S. Burns, G. Borriello, and C. Ebeling, "An FPGA for implementing asynchronous circuits," *IEEE Design & Test of Computers*, vol. 11, no. 3, pp. 60–69, 1994.

[10] G. Taylor, S. Moore, and P. Robinson, "Frequency locked loops," in *7th UK Asynchronous Forum*, Dec. 1999.


## Acknowledgements


The authors would like to acknowledge the support of EPSRC grant GR/L86326, Cambridge Consultants Ltd and AT&T Labs Cambridge and to thank Professor Steve Furber for his discussions relating to this work.