Self Calibrating Clocks for Globally Asynchronous Locally Synchronous Systems

S.W. Moore, G.S. Taylor, P.A. Cunningham, R.D. Mullins and P.Robinson
University of Cambridge, Computer Laboratory,
New Museums Site, Pembroke Street,
Cambridge CB2 3QG, U.K.
Simon.Moore@cl.cam.ac.uk

Abstract

We present a local clocking mechanism based on a tunable delay line which calibrates itself from a low frequency global clock. After initial tuning, the local clock remains calibrated when environmental conditions change. Each module of a large system on a chip can use one of these clock generators running at the optimal frequency for the module.

Communication between locally synchronous blocks is provided by a globally asynchronous interconnect. Reliable low latency communication between the asynchronous interconnect and a local clock domain is made possible by stretching the local clock if a metastable condition could be encountered. Stretching the clock just requires the rising clock edge to be prevented from entering the tuned delay line. Similarly, a sleep state can be entered by stopping the clock and wakeup is almost instantaneous. Fine grained sleeping is possible by sleeping whenever there is no work to be undertaken and waking up as soon as new data appears over the asynchronous interconnect.

1. Introduction

Globally Asynchronous, Locally Synchronous (GALS) systems offer a high performance and low power solution for system on a chip implementors [4]. Clocked building blocks can be integrated onto one chip with independent clocks for each block and an asynchronous interconnect between them.

Synchronising to asynchronous data is a well known problem which can be crudely resolved by latching the data at least twice to allow time for metastability in the latches to resolve. This does not prevent metastability from propagating though the chance is small [5]. A more pressing concern is the latency that is introduced by this scheme.

An alternative strategy is to stretch the clock when there is a risk of metastability [1, 9, 10]. These schemes rely on generating the clock from a delay line because such a clock is simple to stop by gating the clock pulse. Inevitably the performance of a simple delay line is dependent on manufacturing tolerances and environmental conditions. Our experience shows that for most CMOS processes an error of at least 10% can be expected.

In order to provide a far more accurate delay line we propose a run-time calibration mechanism. A low frequency clock (e.g. a 32kHz watch crystal) is used as a reference source and a measurement is made of the number of high frequency clock cycles per low frequency period. The delay line is then adjusted to ensure that it is within 1% of the desired frequency.

2. Overview

An abstraction of the calibratable delay line with GALS interface is shown in Figure 1. An oscillator is formed from a loop consisting of an inverter, arbiter and delay line. The delay line is tuned by a clock calibration circuit — a synchronous finite state machine driven from the low frequency reference clock.

Asynchronous input data arrives in bundled data form, i.e. for each data bus there is an associated request signal which indicates when the data is valid. Bundled data communication structures can provide a reliable asynchronous interconnect between independently clocked synchronous domains. This communications structure is in contrast to Q-modules [7] which sample input signals at regular intervals regardless of whether or not they are changing.

The GALS control block derives data enable signals in response to new data from the asynchronous interfaces. These enables are timed via the clock pause block to ensure that asynchronous data is only sampled when setup violations will not occur. Thus the clock pause block must make a decision about when the next clock edge is likely to occur and prevent data enables from being set

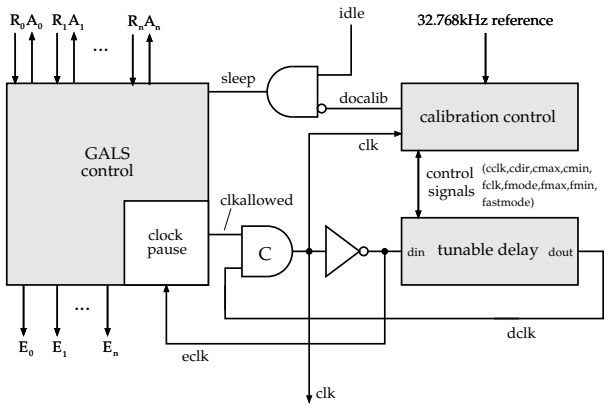

Figure 1. Delay Line with GALS Interface

too close to the clock edge. This circuit relies on Seitz arbiters [8] which may take an unbounded time to resolve if an internal metastable state arises. Fortunately it is easy to delay the clock edge to allow the arbiters time to resolve by keeping `clkallowed` (clock allowed) low. However, during typical operation metastability will not impact on performance as long as the `clkallowed` signal arrives ahead of the `dclk`. The clock pause circuit described in Section 5 achieves this.

3. Tunable delay line

Each cell in the delay line is split into two parts, the delay and completion section, shown in Figure 2, and the control section shown in Figure 3. The control section is further sub-divided providing coarse and fine adjustment of the tunable delay. The number of delay cells required depends on the range of clock frequencies desired, the length of each delay element, the expected range of environmental conditions and the implementation technology.

Operation can be described by considering a transition on the delay input `din`. The transition is delayed by D or by $D + d$ depending on `fout`. Typically d is small, for example, $D/10$. The transition is then directed either to

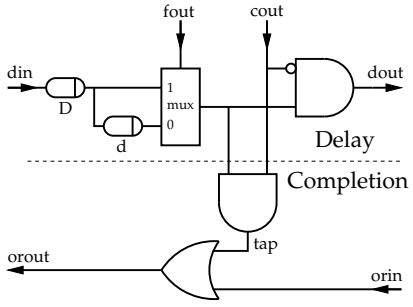

Figure 2. Abstraction of the delay cell

the next delay cell, via `dout`, or into a chain of OR gates, via `tap`, depending on `cout`. The number of delay cells through which the transition passes, encoded in inverse unary on array of `cout` signals, is the coarse setting. Likewise an array of `fout` signals determines the fine setting. The `tap` signals from each cell are ORed together to produce the delay line output. 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.

Although two separate delay elements and a multiplexer are shown, the real implementation is technology specific. In a CMOS implementation with standard cells, the delay can be obtained by one or more complex (slow) gates (Figure 4). Fast/slow selection can obtained by altering an input which affects drive strength thus alters the input-output delay. The corresponding full custom CMOS implementation is a NAND with variable drive strength (Figure 5). The delay of each cell should be such that enough cells are in use to permit the fine control range to be at least that of one coarse step, otherwise there will be gaps in the range of delays possible.

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 low temperature) without increasing the 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. Other schemes are also possible, for example having a fixed number of delay cells and only the fine control mechanism, or using inverters with capacitive loading controlled by a shift register [2].

The control section forms two bidirectional shift registers. The upper shift register provides coarse control. Zeros are shifted in from the left and ones from the right under control of `cdir` and `cclk`. Minimum and

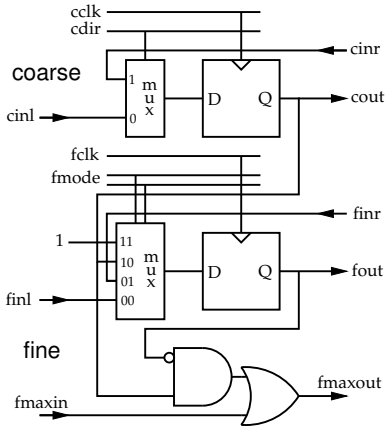

Figure 3. Control section of a delay cell

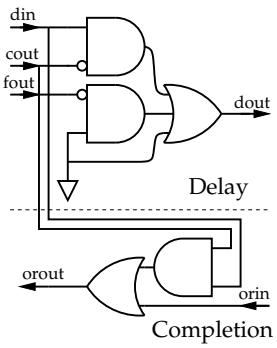

Figure 4. Standard cell delay cell

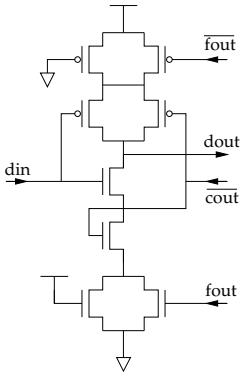

Figure 5. Full custom delay cell

maximum is indicated by the state of the left and right most cells. The lower shift register, controlled by `fmode` and `fclk` provides fine control. Minimum is indicated by the state of the left most cell. Maximum occurs when all of the cells from the left up to and including the cell at which the delay is tapped are at their slow setting and is detected by a second OR structure. The fine setting can be reset to minimum (all 1s) or set to maximum (copied from the coarse shift register).

Adjusting the delay length is a two step process. First consider the case of decreasing the delay. If the fine setting is not at minimum the fine setting can simply be decreased. However, if the fine setting is already at minimum, first the fine setting must be set to maximum and then the coarse setting can be decreased. This mechanism stays "on the safe side"; if the coarse setting were to be decreased first, the clock period might become too short until the fine setting is later re-adjusted. Likewise when increasing the delay, the coarse setting is incremented before the fine setting is decreased.

Normally this adjustment process would occur once per second, frequently enough to track normal environmental changes. However, at initialisation, or during rapid environmental changes it is desirable to calibrate more quickly. A `fastmode` signal, not shown in the diagram, is generated which indicates when the last few adjustments have been in the same direction (increase or decrease) and thus calibration can take place more quickly.

Depending on the delay cell implementation, some assumptions may need to be made to ensure that glitches are not generated and propagated through the delay line when the shift-registers are adjusted. For example, whilst changing the delay selection multiplexer when a transition is travelling through the delay line. In the test implementation, such glitches are too short to be propagated. An alternative solution might use a more complex control structure permitting a new delay length to be selected before the previous one is de-selected.

4. Calibration control

The calibration control module is shown in Figure 6. Calibration is performed by counting how many periods of `clk` occur during one 32kHz reference clock period and then adjusting the tunable delay by one step. Calibration is performed once per second except when fast-mode is active, in which case counting and adjustment takes place repeatedly until calibration is complete.

A synchronous state-machine, shown in Figure 7, clocked from the 32kHz reference 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. A ripple counter (10 bits in the test implementation) is used to count how many `clk` periods occur during the reference period.

The state-machine drives some logic which controls the delay line. `Ssetup` samples the comparator result, `Sclka` and `Sclkb` trigger the two stages of the adjustment process.

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 counting, via the arbiter discussed below. The counter counts the number of positive `clk` edges, until the next state `Swait` reasserts the upper request input to the arbiter. The `Swait` state provides time for the counter and the comparator to stabilize before the `Ssetup` state samples the comparison result and other signals inside the delay line. The comparator result determines if the tunable delay should be increased or decreased. The following `Sclka` and `Sclkb` states then perform the two step adjustment process. Finally another calibration is performed immediately if `fastmode` is set, or if not the state machine waits for the next 1Hz signal. There are several standard synchronous timing assumptions which

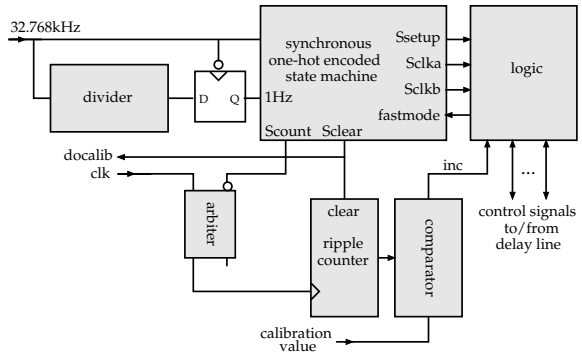

Figure 6. Calibration control module

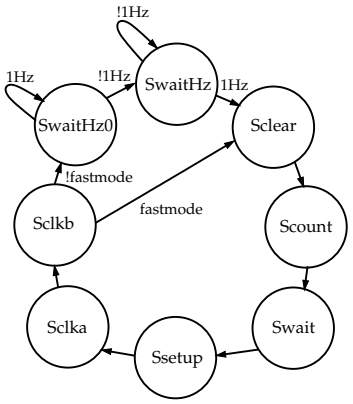

Figure 7. One-hot encoded state machine

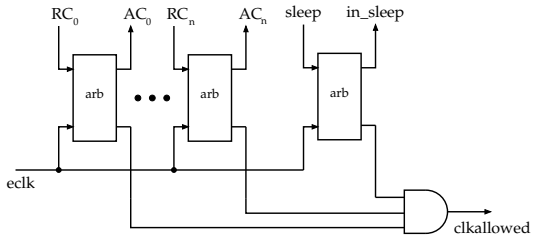

Figure 8. Clock Pause Block

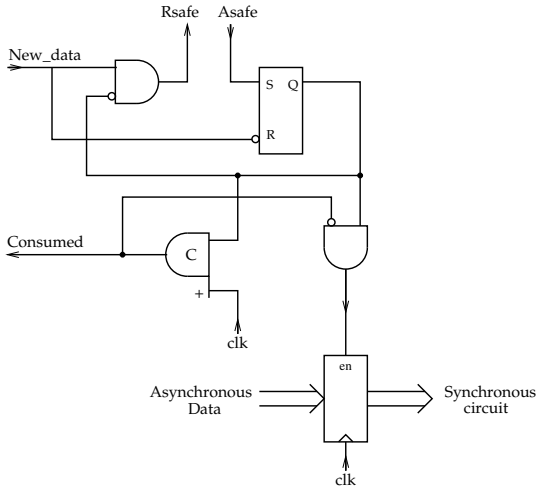

Figure 9. GALS Control Block

are easily met with the low clock period of 30.5μs.

An arbiter with only one output connected is used. 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 30.5μs clock period would be available for the metastability to resolve before the comparator output would be sampled. The arbiter was added to ensure that the counter always receives a proper clock. There is no safe way for the clocked state-machine to wait for the arbiter so the other output is not connected. It is assumed that the arbiter completes within a 30.5μs period. Note that it is necessary to enable/disable `clk` from reaching the counter rather than relying on `docalib` as `clk` might already be running at request of the user.

5. GALS Interface

The GALS interface consists of a GALS control block per asynchronous interface, and one clock pause block. The clock pause block in Figure 8 consists of a number of arbiters, one per GALS control block. Each GALS control block may request that they may present new data to the clocked system by raising $\texttt{RC}_n$. The arbiter will only acknowledge this request (raises $\texttt{AC}_n$) when `eclk` is low. If `eclk` and $\texttt{RC}_n$ rise simultaneously then the arbiter may go metastable internally, but will safely choose one of the requests over the other.

The `eclk` signal predicts when the positive `dclk` signal (from Figure 1) is near and thus when the next positive clock edge needs to be produced. In this example `eclk` predicts that the clock is likely to go high if it is currently low. When `eclk` is high it locks out all of the GALS interfaces by forcing the arbiters to grant in its favour.

When `eclk` is low, it releases its hold on the arbiters giving the GALS control blocks a window of opportunity to supply new data to the synchronous system. The timing of this window can be adjusted by delaying `eclk` so that the window of opportunity is near (but not too close to) the rising clock edge (`clk`). However, if `clk` is to be rarely delayed by the arbitration process then the window of opportunity must not be too close to the rising edge of `dclk`.

A GALS control block is shown in Figure 9. When new data arrives (`New_data+`)[1] a request is made (`Rsafe+`) to the clock pause block which acknowledges with `Asafe+` when it is safe to supply the synchronous circuit with new

[1] `New_data+` means that `New_data` is going high.

data. When Asafe+ is received it sets the RS flip-flop and enables the D-latches so that the new data can be latched on the next clock edge. Shortly after the RS flip-flop is set, the Rsafe- transition will release the arbiter in the clock pause circuit. It is important that Rsafe is high for as short a period as possible in order to minimise the risk of delaying the system clock.

When the synchronous circuit consumes the data on the rising edge of clk, the Consumed signal is raised which lowers the enable to the data-latches. Consumed going high also allows the asynchronous interface to respond by lowering New_data which results in Consumed going low, all ready to receive some more data.

6. Analysis of the GALS interface

To ensure that it is unlikely that the New_data signal will delay the clock we need to test the critical path from New_data followed by eclk going high to the clockallowed signal going high, i.e. how long the GALS interface can delay the clockallowed signal. SPICE simulation of a 0.35μm CMOS implementation indicates that this critical path is 0.98ns provided eclk+ is sufficiently after New_data+ that the arbiter does not go metastable. If metastability does occur then the resolution time depends on the exact timing of the input signals, their rise time and the gain inside the arbiter.

If the tuned delay line were set at 2.5ns (to generate a 200MHz clock) then the GALS interface would be allowed to produce data when the clk is high (for 2.5ns). When clk goes low, arbitration begins, which we know will take 0.98ns. Even if metastability resolution takes no more than 1.5ns, the clock (clk) will not be delayed. In the best case it will take around 0.5 clock cycles to transfer data from the asynchronous interface to the synchronous system, and in the worst case 1.5 clock cycles. If the asynchronous interface is eager (e.g. it is supplying data buffered in a FIFO) then it is able to supply new data every clock cycle.

Metastability can, none the less, cause the tunable delay line to stall from time to time which can interfere with the calibration process. A margin for error may be added to the target delay length to compensate, or two delay lines may be used, one being calibrated whilst the other is in use [3].

The GALS control block is an asynchronous finite state machine. Various internal delay assumptions are made but the external signals are delay insensitive. Analysis of the internal delays (using our in house tool) reveals that all assumptions satisfy the rule "any two gate delays takes longer than any one gate delay". The most critical delay is the D-latch enable signal which must be set before the clock can go high. If necessary an additional delay margin may be added to the feedback from the RS flip-flop to delay Rsafe-.

7. Testing the tunable delay line

A test implementation of the tunable delay line was created for a Xilinx XC4000 FPGA device to see how it would respond to changing environmental conditions. The delay and completion section of each delay cell is implemented by a single CLB. The delay is formed from the CLB and the routing between cells. Fine delay control is achieved by selecting the path the signal takes inside the CLB. The CLB part of the delay is either an H-block or an F-block and H-block in series, the delay input is routed to both F and H blocks. The route to the two blocks is chosen such that the difference between the two delay options is as small as possible. A delay line length of 25 cells was chosen with a target delay of 120ns, which was found to require about 12 delay cells. Note that additional delay exists in the routing outside of the delay line. One simple floor-planning restriction was made to ensure the delay line was in a straight line and thus did not suffer any large jumps between successive steps.

A component not available in the Xilinx architecture is the arbiter. Initially a circuit similar to that in [6] was used. Whilst sufficient for a test implementation, a CMOS implementation would obviously use the cleaner, and faster, Seitz arbiter [8]. However, it was found that the internal clock delay of the Xilinx arbiter circuit added jitter to the clk period. To obtain measurements indicating the accuracy that might be obtained from a CMOS implementation, the arbiter was replaced by an AND gate.

8. Test results

The ability of the delay line to adapt to a changing environment was tested. It is unwise to vary the supply voltage to a Xilinx FPGA because this can corrupt its configuration. Instead the temperature was varied using a hot air gun and freezer spray. The temperature range was approximately -35°C to 60°C. Initially a calibration rate of 1Hz was chosen but this was found to be too infrequent to give good accuracy when under attack by a can of freezer spray and so 4Hz was used. A rate of 1Hz or less should be more than sufficient within a more realistic environment.

Figure 10 shows the results obtained. The graph divides into four chronological sections, initial reset whilst at a maximum temperature, application of freezer spray, no heating or cooling and finally re-heating. As expected at each re-calibration the period is either too long or too short, hence the stepped appearance. The interval between coarse and fine adjustment when coarse adjustment occurs is not exposed in the graph because measurements are only taken preceding calibration. Note the scale of the vertical axis; the error is approximately $\pm 1\%$.

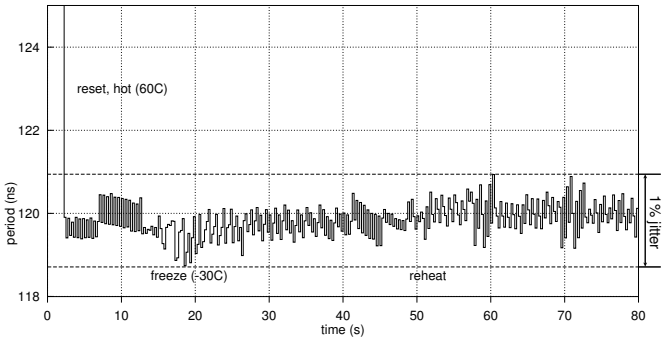

Figure 10. Measured period vs time

9. Analysis of the CMOS tunable delay line

The test results from the Xilinx implementation of the delay line indicate excellent stability despite this implementation being none ideal. Results from SPICE simulation of the 0.35μm CMOS versions (3.3V at 27°C) of the delay cells are given in the table below. The main advantage of the full custom cell is that the course step size can be reduced if a higher frequency clock is required whilst still being able to control the delay to within 2%. None the less, the standard cell variant performs well and is appropriate for some applications.

	fine step size	**coarse step size**
standard cell	0.065ns	0.865ns
full custom	0.059ns	0.604ns

10. Conclusion

This paper has presented a tunable delay line design which can be used to provide local clocks in a globally asynchronous locally synchronous (GALS) environment. Experimental and simulation results indicate that the tunable delay line can provide a highly stable clock (jitter is around 1% after initial calibration) despite environmental changes. Stability is similar to many PLL designs but the tunable delay line is fully digital and can be constructed from standard cells. Furthermore, the tunable delay line can the stopped within one clock period and restarted instantaneously which ideal if one wishes to sleep to save power whenever possible. For example, when the synchronous system is waiting for data from the asynchronous interconnect it is trivial to stop the clock and restart it when data arrives.

The paper also presented a GALS design to interface an asynchronous interconnect to the locally synchronous environment. Clock stretching is used to prevent metastability when the synchronous system samples data from the asynchronous interconnect. However, unlike other designs in the literature, our GALS interface is capable of granting many asynchronous data requests in parallel. Furthermore, arbitration between the asynchronous and the synchronous sides is undertaken in advance of the next positive clock edge. This ensures that the clock is almost never delayed so the synchronous system runs at its full rate.

References

[1] David S. Bormann and Peter Y. K. Cheung. Asynchronous wrapper for heterogeneous systems. In *Proc. International Conf. Computer Design (ICCD)*, October 1997.

[2] G. Geannopoulos and X. Dai. An adaptive digital deskewing circuit for clock distribution networks. In *International Solid State Circuits Conference*, 1998.

[3] George Taylor, Simon Moore, Steev Wilcox and Peter Robinson. An on-chip dynamically recalibrated delay line for embedded self-timed systems. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, April 2000.

[4] A. Hemani, T. Meincke, S. Kumar, A. Postula, T. Olsson, P. Nilsson, J. Öberg, P. Ellervee, and D. Lundqvist. Lowering power consumption in clock by using globally asynchronous, locally synchronous design style. In *Proc. ACM/IEEE Design Automation Conference*, 1999.

[5] Howard W. Johnson and Martin Graham. *High-Speed Digital Design — A Handbook of Black Magic*. Prentice Hall, 1993.

[6] Simon W. Moore and Peter Robinson. Rapid prototyping of self-timed circuits. In *Proc. International Conf. Computer Design (ICCD)*, October 1998.

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

[8] Charles L. Seitz. System timing. In Carver A. Mead and Lynn A. Conway, editors, *Introduction to VLSI Systems*, chapter 7. Addison-Wesley, 1980.

[9] Allen E. Sjogren and Chris J. Myers. Interfacing synchronous and asynchronous modules within a high-speed pipeline. In *Advanced Research in VLSI*, pages 47–61, September 1997.

[10] K.Y. Yun and A. E. Dooply. Pausible clocking based heterogeneous systems. *IEEE Transactions on VLSI Systems*, 7(4):482–487, December 1999.

Acknowledgements

The authors would like to acknowledge the support of EPSRC grant GR/L86326, Cambridge Consultants Ltd and AT&T Labs Cambridge.