Stopping the Clock: Gating vs Pausing

Andrew Royal and Peter Cheung,
Dept. of Electrical and Electronic Engineering, Imperial College, London

Abstract— **The aim of this work is to investigate the advantages and disadvantages of two techniques for stopping the clock of a synchronous design: clock gating and clock pausing. In particular, the stability of the derived clock is discussed. The merits of clock pausing when used in a globally asynchronous locally synchronous system are also studied at length.**

I. Introduction

In a synchronous system, power is often wasted by redundant switching. There are many registers in such a system and even if the inputs to the registers do not change or the registers are disabled, some internal switching may still occur. Obviously it would be beneficial to reduce or eliminate this if possible.

A common approach is to stop the clock. If a register does not see a clock edge, it will not trigger and there will not be any internal switching. However, interfering with the clock is often a very contentious issue as synchronous designers like to have a stable and reliable clock signal for their designs. Nevertheless, there are two common method for stopping the clock.

The first of these methods is clock gating. A global clock is distributed across the system and a logic gate will block it at some level to prevent it from progressing to lower levels of the clock tree. Hence the clock is prevented from reaching the registers.

A second method is known as clock pausing. Rather than use a crystal oscillator for the clock, a more controllable ring oscillator is used. Here, a gate is inserted into the oscillator ring itself to directly stop the clock from producing the next (rising) edge. This method is commonly used in globally asynchronous locally synchronous systems.

II. Clock Gating

A rather crude manner of stopping the clock to a module is to use clock gating. Here, a logic gate is used to block the clock before it gets to its target registers, hence power is not wasted in redundant clocking. The basic scheme is illustrated in figure 1.

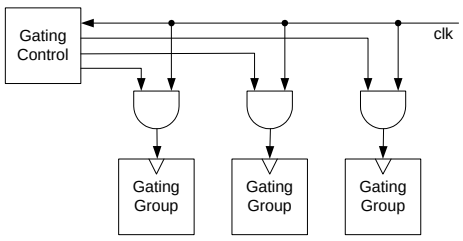

Fig. 1. Clock gating

There are some inherent problems with adding gates into the clock tree. Additional delay is introduced into the clock lines which can worsen clock skew. It can also produce glitches in the clock tree which could have potentially disastrous implications for the stability of the circuit.

A circuit designer is usually only concerned with the gate level operation of the circuit and will not consider clock distribution which is added at the layout stage. If the designer adds clock gating at the netlist stage, this may not be taken into account by the clock tree added at a later stage in the design process. If clock gating is used, the gating strategy should be developed as part of the clock distribution tree. Hence the delay introduced by the gates can be taken into consideration so that the delay in the clock tree remains constant in all branches of the tree and so clock skew can be kept at a minimum.

Control of the gates must also be considered. Gating control is generally synchronous. This is essential, so that the clock is not enabled or disabled in a critical phase of the clock. If the gated clock is re-enabled before the global clock goes into the low phase of the cycle, the gated clock may immediately go high, causing registers to trigger and causing a shorter than normal clock period which may cause stability issues. The clock enable signal must usually be glitch free, or again the gated clock may go high unexpectedly and trigger registers prematurely. Glitches may be allowable if care is taken to ensure that they only occur when the glitches will not propagate through to the clock. For example, if a clock is gated with an AND gate, glitches on the control input may be allowable while the input clock is low. Clock gating is likely to be at a some intermediate level in the clock tree, hence the gates must be sized to be able to drive all the branches further down in the tree. Because of this, the drive capability of the gating control signal must be considered. This may have further implications, since any buffering in the control path will add further delay.

It has been observed that the addition of clock gating consumes additional power. [3] Additional gates will not only consume more power but also take up chip area and require extra wiring, both of which are likely to increase the complexity of routing. This extra power spent may, in extreme cases, outweigh the power conserved by preventing the registers on the gated branches of the clock tree for switching. Because of this, it is important to ensure that there are not too many gating signals which will require excessive routing and that the gating signals do not switch too frequently. Thus it is particularly important to choose clock gating groups carefully, for example using the methods described in [10] or [3].

It may also be possible to use clock gating at a very low

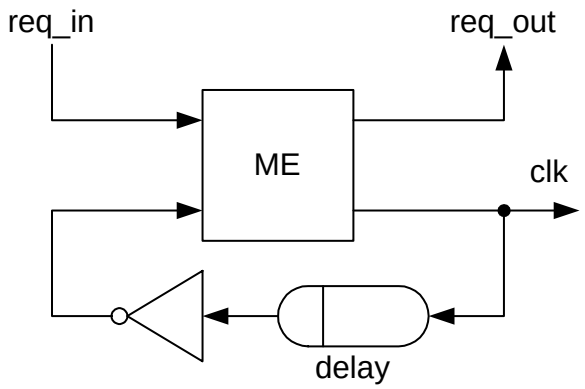

Fig. 2. Simple clock stretching circuit

level, exploiting the fact that when a flip-flop is not switching it's input is irrelevant, as described in [12]. This allows the logic function for the input to be simplified and so long as the gated clock signal is also simple it can dramatically reduce power consumption. However, additional skew may be added to the clock signals for individual registers and since this is at such a low level great care must be taken to consider the skew relative to adjacent components. Hence the composability of components designed in this manner is questionable.

III. Pausable Clocks

Crystal oscillators are widely used because their oscillations are very stable and at a constant frequency. However, this stability is not always an advantage: crystal oscillators cannot be easily paused and the frequency cannot be easily changed. It may still be possible to gate the clock or derive a clock of a different frequency using a clock multiplier. A phase-locked loop could conceivably be used, though such an analogue device does not integrate well into an inherently noisy digital system.

A simple alternative is an inverter ring oscillator. This exploits the delay in digital inverters by cascading them into a loop. So long as there is an odd number of inverters, the nodes between inverters will oscillate between the '0' and '1' states. One of these nodes can then be tapped as a clock. A simple pausable ring oscillator is shown in Figure 2. Although the delay varies as the technology is scaled, it will not necessarily vary at the same rate as other parts of the design as wire delay becomes more significant, which will effect different parts of the design by differing degrees. Also, as we expect process variations in any CMOS design, the frequency of oscillation cannot be accurately set with a simple inverter ring. There are methods for tuning the delay of such a ring so the frequency of oscillation can be varied, for example [6] or [8], though in both of these cases it may take some time to calibrate the clock before it can be used. The clock can also be stopped completely by adding a gate into the ring. This can be used for clock stretching/pausing.

It is possible that when the clock is paused, there will be problems starting it again. Ring oscillators are generally considered to be digital devices, generating a perfect square wave. However, they really should be treated as analogue oscillators, since the output of a gate starts changing the instant its input starts to change, not when at a distinct threshold. In an analogue oscillator, the loop gain must be exactly -1 to ensure stable oscillation. This operating point may take some time to obtain. Before stable oscillation is obtained, the amplitude and frequency of oscillation may change. However, the self calibrating clock in [6] is capable of starting again immediately. In [13], a trace of the clock is shown and it appears restart after pausing. However, in this example the pause is only for a short period of time (less than a cycle) and the oscillation does appear a little erratic after the pause, even though only one more cycle is shown before the trace ends.

IV. Globally Asynchronous Locally Synchronous (GALS) Systems

Rather than force everything in a digital circuit to be synchronised to the same clock, it may be preferable to divide the design into blocks which are internally synchronous but oblivious to the clocking of other blocks. An asynchronous handshake protocol is then used to transfer bundled data (i.e. all data always arrives before the associated request) between modules. Hence the module can be said to be locally synchronous while the system as a whole is globally asynchronous. Early examples include [9] and [2]. It can immediately be seen how this alleviates the problem of clock skew, since only the skew occurring inside each block need be considered. However, some additional circuitry will generally be required to synchronise data entering each module, which in itself will consume power and increase circuit area.

Because a GALS system contains asynchronous data channels which enter synchronous modules, metastability may occur in the input ports of the modules. Metastability is phenomenon commonly found in bistable components such as latches and arbiters when data in asynchronous. While internal switching is occurring, changes to the input data can cause the internal nodes to deviate from their path towards a resolved state. If the change occurs at a specific point along this path, the nodes can go towards a third stable state, known as the metastable state, which is between resolved states. Though the probability of actually hitting this point is zero, the closer the input change is to this point, the longer the device will take to return to a resolved state. This will tend to cause problems at later stages in the circuit, which will not see a resolved value at their input and hence may be unable to produce a resolved output. The general theory of metastability is described in [5].

Two different approaches have been used to combat metastability in a GALS system. The first is to use some kind of detector to sense when the input to a register is unresolved. If so, the clock is paused or stretched to delay the next rising edge until the metastability has been resolved. This scheme is used in [11].

Alternatively, metastability can be avoided by using mu-

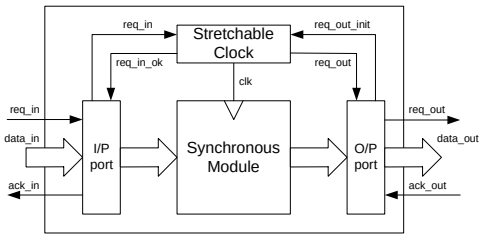

Fig. 3. GALS module with clock stretching

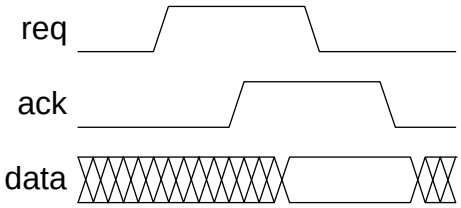

Fig. 4. Late data valid protocol. The initiator sends a request and waits for an acknowledge from the receiver before asserting the data and then removing the request. Data must be held until after the receiver removes the acknowledge.

tual exclusion to arbitrate between asynchronous requests to the module and the clock. Locally generated clocks made from inverter ring oscillators are used since it is easy to control and pause this type of clock. When the request locks out the arbiter, the clock is paused. This scheme appears to be more widely used, for example in [13] or [7]. A module using the scheme is shown in Figure 3. A 4-phase late data valid handshaking scheme is asserted, as shown in Figure 4. Under this scheme a request is send out before any data is transmitted. Once an acknowledge signal has been received, data is transmitted and the request is removed. However, the data must be held until after the acknowledge has also been removed. Such a scheme can be implemented with asynchronous state machines in the input and output ports. This protocol ensures that the receiver has time to create a lock-out on the clock signal before data is transmitted, thus theoretically eliminating the possibility of metastability occurring. However, a mutual exclusion element is itself prone to metastability, albeit with a greatly reduced probability. Hence it may not be possible to completely eliminate all possibility of metastability. Note that with this scheme, modules will only necessarily be paused when data is being transmitted from another module.

Note that there are no metastability problems when data leaves the synchronous module and enters the asynchronous environment. However, the output port may need to pause the clock if the receiving module is taking a long time to complete the handshake.

A slightly different scheme is that used in [1]. Here, a mutual exclusion element is not used. Instead, a module has two port select signals which go to its input and output ports. These indicate that the module needs data or has data to send respectively. Transitions occur soon after a clock edge and the corresponding port(s) will immediately send a clock stretch signal. An active port will then send out a request, while a passive port will wait for a request to indicate that new data is available or that the receiver needs data. This scheme is shown in figure 5. The advantage of this scheme is that the clock stretch signal is effectively synchronous, as it occurs as a direct result of the synchronous port select signal, hence the possibility of metastability is eliminated from the clock stretching circuit. However, unless the asynchronous state machine in the port is carefully designed, it may be possible for metastability to arise there. The clock will remain paused from the cycle a port is selected to the cycle the corresponding handshake is complete. Throughput will be limited to the slowest module, since surrounding modules will be waiting to send data to or receive data from that module. Compare this to the schemes using mutual exclusion, which allow clocking even when no data is supplied/consumed. One could argue that there is usually no point in the module clocking when there is no data being supplied or there is nowhere for new data to go. An additional data valid signal may be required in this case.

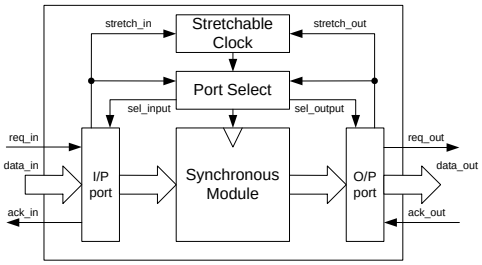

Fig. 5. GALS module with port select

The scheme described above assumes an active output port and a passive input port. However, the operation of the port select implies that the module actively tells its import port to receive data. Hence it may make more sense to have both an active output and an active input. A passivator will allow two active ports to communicate by waiting for a request from each before sending acknowledge signals back to both. Again, since the request signals will be sent out by the synchronous module, metastability problems should be eliminated.

An additional benefit of using GALS with clock pausing is that the locally generated clock can be paused not just when an asynchronous request is initiated, but also when the module is not doing any useful calculations. This can greatly reduce redundant switching and precharging and hence power consumption. Also, existing synchronous modules may become GALS modules by simply adding an asynchronous wrapper [1]. Existing synchronous design methods may be used in designing the module, with the asynchronous protocol only added at the top level.

One drawback of a GALS system is that additional circuitry is required for every module in the design. At the very least, the module needs input and output ports to implement the asynchronous protocol between modules. In most cases, a clock multiplier or locally generated clock is

used. Both of these will consume additional power, though hopefully this will be insignificant compared to the potential power saving. Because of the asynchronous protocol, the designer must be wary of deadlock, i.e. a request may never be acknowledged because it is waiting for acknowledgement from another module which itself is depending on the first handshake completing. Multiple clock domains may not be practical for some applications. For example, if data is produced by one module and consumed by the next on every clock cycle, we can intuitively see that they should run at the same speed. Hence multiple clock domains are not useful for such an application. However, a GALS scheme is much more useful for circuits where data is rarely or sporadically transferred between modules. Also, it may be possible to have a single unit constantly producing data, but have multiple modules consuming that data, each at slower rates which sum to the rate of data production [4].

V. Conclusion

Both clock gating and clock pausing as part of a globally asynchronous locally synchronous system are viable methods of stopping the clock to save power. Both have their drawbacks and both have their distinct advantages.

One must consider the problems with stability of the clock. Clock gating is prone to causing glitches on the clock, though through careful design this effect can be minimised. Similarly, there may be a possibility of metastability on a clock which is paused with a mutual exclusion element, though again this can be avoided with careful design of the input ports.

Another import design aspect is that of clock routing. In a GALS system where purely local clock generation is used, there is obviously a cut on clock routing. However, if the local clock requires a clock reference, a global clock must still be distributed. There will still be a clock distribution advantage though, as the global clock does not need huge buffers to drive local clock trees. Local clock trees will still exist, but the global clock need only be used to drive the calibration circuit. Also note that generally the local clock is calibrated by counting the number of pulses it generates in one pulse of the global clock, so the global clock will be at a low frequency in this case. Compare this to a global clock in a clock gating scheme, which needs to run at the same rate as the individual modules and needs enough buffering to drive all modules, since we cannot discount that they will all be enabled at the same time. Clock skew may occur globally, while with GALS we only need worry about the local clock skew.

A globally asynchronous locally synchronous scheme also has the advantage of allowing multiple clock domains and the integration of asynchronous circuits. Allowing a different clock rate for each module allows modules to be independently optimised, but if a module produces data every cycle which is passed to another module consuming data every cycle, there is little point in them running at different rates. Some modules may be better designed using asynchronous techniques, with this scheme we can choose the best paradigm for each module.

However, despite the advantages of GALS, clock gating may still be useful. It is very simple because it only requires logical gates. Some implementations of GALS require components such as arbiters or C-elements, which are not included in some design libraries. A GALS system requires additional circuitry for each asynchronous wrapper, although this may be quite small. Clock gating may require additional gates, but this will usually be in lieu of clock buffers. Hence the final decision between the two approaches is close run. If only a single clock domain is required, particularly if modules consume and produce data every cycle or a constant throughput is necessary, and methods to eliminate clock glitches are implemented, or if even a low probability of internal metastability is unacceptable, clock gating is the better approach. If throughput need not be optimal and multiple clock domains are imposed, the designer wishes to exploit multiple clock domains or the design is large and fast enough that clock skew may become significant, GALS should be used.

Acknowledgment

The author was sponsored by LSI Logic.

References

[1] D. S. Bormann and P. Y. K. Cheung, *Asynchronous wrapper for heterogeneous systems*, Proceedings of the International Conference on Computer Design (ICCD) (1997), 307–314.

[2] Daniel M. Chapiro, *Globally synchronous locally synchronous systems*, Ph.D. thesis, Stanford University, October 1984.

[3] D. Garrett, M. Stan, and A. Dean, *Challenges in clockgating for a low power asic methodology*, Proceedings of the International Symposium on Low Power ASIC Methodology (1999), 176–181.

[4] P. Liljeberg, J. Plosila, and J. Isoaho, *Asynchronous interface for locally clocked modules in ulsi systems*, The IEEE International Symposium on Circuits and Systems (ISCAS) **4** (2001), 170–173.

[5] L. R. Marino, *General theory of metastable operation*, IEEE Transactions on Computers **C-30** (1981), 107–115.

[6] S. W. Moore, G. S. Taylor, P. A. Cunningham, R. D. Mullins, and P. Robinson, *Self-calibrating clocks for globally asynchronous locally synchronous systems*, Proceedings of the International Conference on Computer Design (ICCD) (2000), 37–78.

[7] J. Muttersbach, T. Villiger, H. Kaeslin, N. Felber, and W. Fichter, *Globally-asynchronous locally-synchronous architectures to simplify the design of on-chip systems*, Proceedings of the 12th Annual IEEE ASIC/SOC Conference (1999), 317–321.

[8] T. Olsson, P. Nilsson, T. Meincke, A. Hemam, and M. Tokelson, *A digitally controlled low-power clock multiplier for globally asynchronous locally synchronous designs*, The IEEE International Symposium on Circuits and Systems (ISCAS) **3** (2000), 13–16.

[9] Miroslav Pěchouček, *Anomalous response times of input sychronisers*, IEEE Transactions on Computers **C-25** (1976), 133–139.

[10] G. E. Téllez, A. Farrahi, and M. Sarrafzadeh, *Activity-driven clock design for low power circuits*, IEEE/ACM International Conference on Computer-Aided Design (2000), 62–65.

[11] W. S. VanScheik and R. F. Tinder, *High speed externally asynchronous/internally clocked systems*, IEEE Transactions on Computers **46** (1997), no. 7, 824–829.

[12] Qing Wu, Massoud Pedram, and Xunwei Wu, *Clock-gating and its application to low power design of sequential circuits*, IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications.

[13] Kenneth Y. Yun and Ryan P. Donohue, *Pausible clocking: A first step toward heterogeneous systems*, Proceedings of the International Conference on VLSI in Computers and Processors (1996), 118–123.