

GALS

Globally Asynchronous, Locally Synchronous Systems

Most digital circuits are synchronous. Each processing block (e.g. adder, counter, multiplier) has registers at its inputs and outputs, with data arriving at the inputs on the rising edge of the clock, propagating through the block to arrive at the outputs before the subsequent rising clock edge, when it is clocked through to the next block. We must ensure that the data going into the output register has stabilised before we clock it through to the output, hence the time it takes to propagate across the block determines the minimum clock period for that block. When single blocks are built up into a module (e.g. transform engine, filter, pre-parser) the clock speed of the module is determined by the worst case delay, i.e. the slowest component block, usually with some additional slack for safety.

In a conventional, globally synchronous system (e.g. MPEG decoder, microprocessor), a clock signal from a single source must be distributed to every registered block in the system. The clock is distributed in a tree of buffers to spread across the whole system. In this way, all blocks are synchronised, with data ideally arriving from all different sources at the same time.

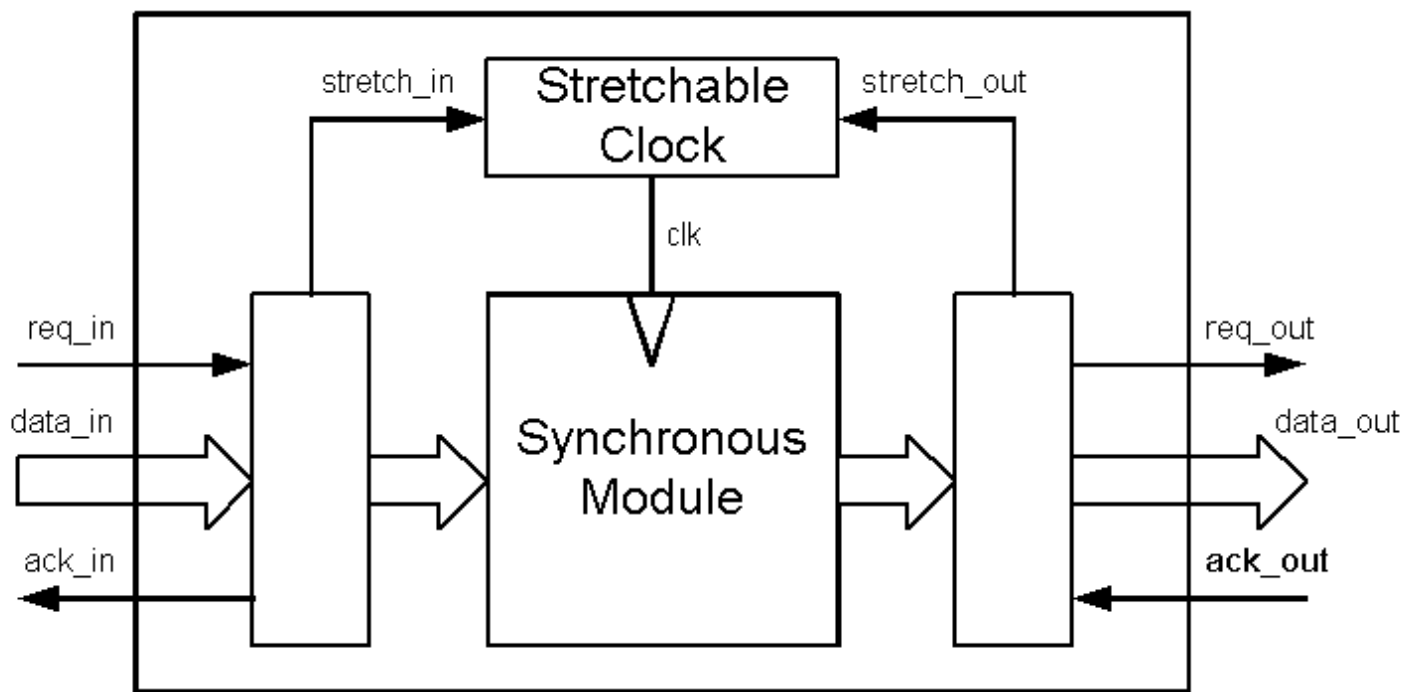
The first problem with this is one of clock skew. Unless the designer is extremely careful, the delay between the clock input and any two blocks in the system may be quite different. Hence they will be out of synch, which could cause problems if the data from those two block converges in another block.

Another problem is power consumption. Distributing and buffering the clock to so many destinations consumes a tremendous amount of power, maybe as much as 40% of the total power consumed in the system. Additionally, when registers or even entire modules are disabled, the clock will still be wastefully distributed to them.

Finally, in a large modern system, many component modules are combined to build up the system. Some may be standard modules from a library, some may be specifically designed for the system. This may mean that modules have been designed to run at different clock speeds. In this case, we can either set the clock to the slowest clock speed required or have multiple clocks. A module to receive a bitstream may be fixed to a particular clock speed by the specification of the bitstream, in which case we may be forced into using multiple clocks. However, synchronisers between clock domains are notoriously messy, may be prone to metastability and introduce large latencies.

GALS systems aim to combat all three of these problems by doing away with a global clock and instead having a locally generated clock for every module. It can be seen how this reduces the complexity of clock distribution since the clock need only be locally distributed. Obviously, modules cannot be synchronised if they all use different clocks, so we use an asynchronous protocol to communicate between them. Leaving a synchronous block into the asynchronous domain is a reasonably trivial matter. However, re-entering another synchronous block is problematic.

We cannot predict exactly when data will arrive from the asynchronous domain, so it is entirely possible that it will come very close to the clock edge. If this happens, it may cause metastability: the register output may take an unbounded length of time to settle. To avoid this, GALS systems stretch the clock, i.e. they prevent the clock from rising if asynchronous data is arriving. Hence a single module in a GALS system generally looks like this:



A lot of research has already been done into the stretchable clock and protocols for the input and output ports. We now aim to improve methodologies

The following people are currently involved in GALS research:



Andrew Royal
PhD Student, Imperial College



Peter Cheung
Reader In Digital Systems, Dept. of EE, Imperial College
Academic Supervisor

Publications

None at present

Last updated: 27th April 2001