Nexus: An Asynchronous Crossbar Interconnect for Synchronous System-on-Chip Designs

Andrew Lines
Fulcrum Microsystems
26775 Malibu Hills Road, Calabasas, CA 91301
lines@fulcrummicro.com

Abstract

Asynchronous circuits can provide an elegant and high performance interconnect solution for synchronous system-on-chip (SoC) designs with multiple clock domains. This "globally asynchronous, locally synchronous" (GALS) approach simplifies global timing and synchronization problems, improving performance, reliability, and development time.

Fulcrum Microsystems' SoC interconnect, "Nexus", includes a 16 port, 36 bit asynchronous crossbar which connects via asynchronous channels to clock domain converters for each synchronous module. Each synchronous module has its own local clock domain, and can send a variable length burst of data to any other module.

In TSMC's $130nm$ LV, Low-K process, the system achieves $1.35GHz$ at $1.2V$ with less than $5mm^2$ area. Power scales linearly with bandwidth, from a few mW of leakage to $8W$ at the peak $780Gb/s$ cross-section bandwidth. Latency through the interconnect is $2ns$ plus $\frac{1}{2}$ to $\frac{3}{2}$ clock cycles of the receiving module.

This compares favorably with other SoC interconnect solutions that have less bandwidth, higher energy per transfer, and longer latencies. Nexus is an innovative and comprehensive solution to the challenge of SoC interconnect.

1. Introduction

System-on-Chip design implies the integration of many different components on the same chip. Due to the wide variety of cores and I/O interfaces found on a typical SoC, it is usually infeasible to maintain a single global clock. Communication between clock domains must then be carefully synchronized. Metastability is introduced, along with some uncertainty in timing. Essentially, any chip with multiple clock domains is globally "asynchronous".

While many techniques have been developed to handle the integration of multiple clock domains, most rely on a localized clock-domain-crossing circuit which lets one clock domain talk directly to another. Long range communication across the chip is still done using synchronous circuits, which require at least one widely distributed clock. This is difficult to do at high frequency, usually resulting in a slower and wider interconnect.

We present a more elegant and efficient solution to the multiple clock domain problem. Instead of gluing synchronous domains directly to each other with clock domain bridges, we use asynchronous circuit design techniques to handle all clock domain crossing as well as all cross-chip communication and routing. This allows all synchronous modules to have their own clock, without introducing any frequency or phase constraints between them. The PLL and clock distribution can be entirely local to each synchronous core, easing timing closure and improving the reusability of cores across multiple designs.

The GALS approach has been studied for many years in the academic community. Recent publications describe support for clock boundary crossing with pauseable clocks [1] and the use of delay-insensitive 1of4 channels and routing primitives [2]. Nexus is unique as a complete, verified, optimized, and practical solution for SoC interconnect that will soon appear in several commercial chips.

2. Asynchronous Integrated Pipelining

Since "asynchronous design" is defined merely by the absence of a clock, there are many possible styles and approaches. The particular style used by Fulcrum is based on the quasi-delay-insensitive (QDI) timing model [3]. This requires that the circuit functions correctly regardless of the delays of all gates or most wires.[1] This is a very conserva-

[1] Some "isochronic" forks in wires must have bounded relative delays. In our designs these are local to small cells and quite safe.

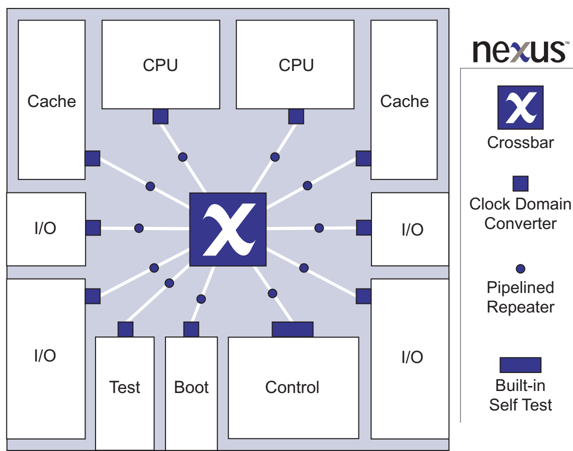

Figure 1. Typical SoC using Nexus.

tive model, since it forbids all forms of timing races, delay assumptions, glitches, and even clocks. This timing model is particularly attractive for SoC interconnect, because QDI circuits will work robustly over huge delay variations caused by power supply droop, in-die variation, local heating, and crosstalk.

In a QDI system, a separate wire can't be used to indicate when a data wire is valid, since you can't make an assumption about the relative delay of the wires. Instead, the data value and validity are mixed onto two wires (dual-rail or 1of2). A backward-going acknowledge wire is included for flow control. These wires together form an asynchronous "channel". When both data wires are 0, the channel is considered "neutral" and no data is present. To send a bit, the sender raises either the 0 data rail or the 1 data rail to send a logical 0 or 1. Once the receiver has received and stored the data, it raises the acknowledge wire. Eventually the sender will set the data rails back to neutral, and the receiver will then lower the acknowledge. This is called a 4-phase dual-rail handshake. To reduce power, we usually use a 1of4 encoding instead of 1of2.

Fulcrum's circuits use precharge domino logic plus some control overhead which combines logic with pipelining. This is smaller and faster than previous QDI approaches. The forward datapath is similar to that of full-custom synchronous designs, like the Pentium4, except without race conditions or explicit latches. A typical pipeline stage has 2 transitions forward latency and an 18 transition cycle time. This design style was originally developed at Caltech from 1995-1999 [4, 5].

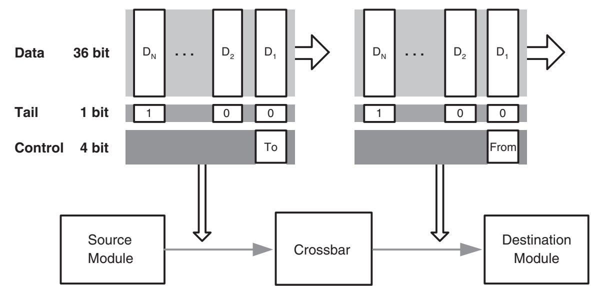

Figure 2. Burst format.

3. Nexus Architecture

Nexus is a crossbar-based interconnect which connects several locally synchronous modules to each other on the same chip. Each module may have an independent clock frequency or phase. Clock domain converters bridge the asynchronous interconnect to each synchronous module. Asynchronous channels carry the data across the chip to and from the central crossbar. The crossbar includes routing and arbitration circuitry to resolve contention on output ports. All parts of the system are safely flow controlled. A typical system configuration is shown in Figure 1.

Nexus relies on one-way transfers of data, called "bursts". A burst contains a variable number of data words, terminated by a tail bit. Each burst is routed with a sideband TO channel, which is converted into a FROM channel when it leaves the crossbar. Bursts are routed atomically and cannot be fragmented, interleaved, duplicated, or dropped. Links are created when the TO control arrives at the crossbar and wins the arbitration, and closed automatically when the last word of the burst exits the crossbar. The burst format is shown in Figure 2.

In the asynchronous portions, channels are encoded as bundles of 1of4 data rails plus acknowledges. On the synchronous interfaces, channels use a simple request/grant fifo protocol. On the rising edge of the clock, if the sender's Request line is asserted and the receiver's Grant line is asserted, then the data is advanced. Either the sender or receiver can stall the transfer. This is functionally equivalent to the asynchronous flow control. All channels are unidirectional, so every module has both an ingress and egress channel.

Round trip transactions can be implemented as split-transactions, with a request burst going one way and a completion burst returning. Various ordering properties, such as producer-consumer and global store ordering are preserved by the system. Thus, many legacy bus protocols can be tunneled through Nexus, with substantial improvement in performance due to the elimination of bus contention.

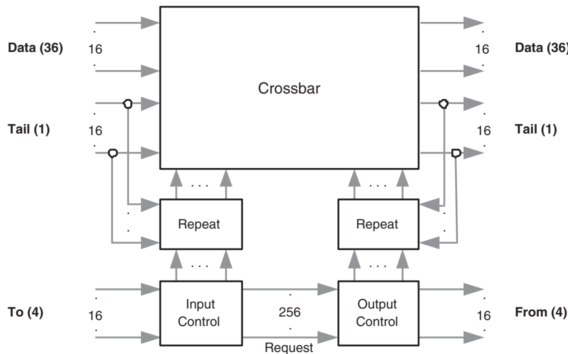

Figure 3. Crossbar decomposition.

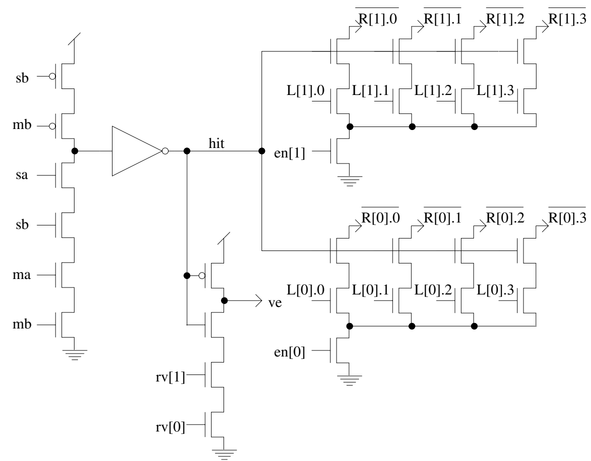

Figure 4. Crossbar datapath grid circuit for 16 ports and 4 bits.

The existing versions of Nexus support 16 modules, a 36 bit datapath plus the tail bit, and 4 bit TO/FROM channels. All subsequent descriptions refer to this configuration, although many variations are possible.

4. Crossbar

The crossbar component of Nexus is implemented by decomposing it into smaller circuits which communicate on internal channels. The largest part is the datapath of the crossbar, which is responsible for muxing all input data channels to all output data channels. It is controlled by a "split" control channel S[i] for each input which says which output to send a burst to. It also requires a "merge" control channel M[j] for each output which says which input to receive a burst from. The split control comes from an input control block for that port, which also receives TO. The merge control comes from an output control block for that port, which also sends FROM. In between the input/output control and the datapath are repeat circuits which replicate the same split/merge control until a tail bit of 1 passes through the link, thus routing a variable length burst. The organization of these components is shown in Figure 3.

4.1. Datapath

The heart of the crossbar is a 16x16, 4 bit demux-mux grid. Each input port broadcasts its split control and input data across a row. Each output port broadcasts its merge control and collects its output data on a bus for each column.

The 4-bit split and merge control are both encoded as two 1of4 codes. A "hit" circuit for each grid point checks that both the split and merge control have selected it. The hit signal enables the data to be transferred from the input bus (L) to the inverted output bus (_R). After the data is latched on the output bus, an acknowledge signal is routed backward through the same grid point. Then both sides complete the 4-phase handshake and return data and then acknowledges to their neutral state. The circuit for the central crossbar grid point is shown in Figure 4.

These 16x16 4 bit crossbars are then stacked to compose a 36 bit datapath. The split and merge control are distributed to each of the 9 chunks with some amount of pipelining. Therefore, there are no performance bottlenecks associated with extending the datapath, and a 128 bit crossbar would be just as high frequency, with only a slight increase in latency. 4 bit crossbars were chosen to minimize the wiring bandwidth in the layout.

4.2. Repeat Until Tail

There is a similar 1-bit crossbar for the tail bit, except it also makes a copy of the incoming and outgoing tail bits. These copies go down to "repeat" units on each split/merge control channel which will repeat the same control values for each word of a burst until the tail is 1. This takes care of routing variable length bursts atomically, and the rest of the crossbar need not be aware of the length of the burst.

4.3. Input Control

Each input port has an input control unit which receives the TO and copies it to the S control. It also sends a token on a request channel (1 data rail and 1 acknowledge) to the selected output control unit. There are 256 request channels connecting all 16 input control units to all 16 output control units.

4.4. Output Control

The output control unit waits until it receives a request from one or more input ports. It picks the first one and sends its port number on FROM and M. If multiple requests arrive at exactly the same time, metastability occurs and a metastability filter is included to wait for the metastability to resolve. Note that in a QDI circuit, such metastability introduces only minor uncertainty in the latency of arbitration, not a chance of failure. It is the fixed clock period that makes metastability a source of failures in synchronous designs.

If an input port were able to make two requests to two output ports at once, it might win its second request first. If another input had also requested the same output ports but in the opposite order, it too could have its second request win first. Thus the two inputs would each have won permission to send to their second destination, but they have data waiting on the input ports intended for their first. This results in deadlock.

There are two ways of solving this problem, both of which essentially require that each input have at most one request "outstanding" at a time; that is, it must be certain it has won its first arbitration before starting a second. This can be done by introducing backward flowing grant channels from the outputs to the inputs. A trickier approach is to remove enough of the integrated pipelining until the first request token blocks the second until it has won the arbiter. The first approach was used in our early crossbar designs, but the second is used in current designs since it is substantially smaller. This constraint introduces a performance bottleneck in the control circuitry which causes it to be slower than the rest of the system, by 25% or 100% for the two approaches. However, since arbitration is also pipelined and occurs only once per burst and and not once per word, this bottleneck rarely affects performance, and never matters for bursts of 2 words or longer.

4.5. Comparison to Synchronous Crossbars

Synchronous crossbars can use a similar datapath grid to transfer data from all inputs to all outputs. The input split control and data would be broadcast across the row. The merge control would be broadcast up the column and combined in a hit circuit. An OR tree would be used to merge up the output data. Or in a full custom flow, a dynamic logic bus could also be used. Input and output control circuits could be similar. Extra support for flow control would also be needed. Careful floorplanning is necessary to make sure the N^2 intermediate links are very short.

However, the asynchronous design can trivially support links at all different frequencies, from DC to the maximum. It has no timing assumptions to verify, and operates correctly over a wide voltage and temperature range. It also only consumes power for bandwidth that is actually used. A synchronous design latching 16 inputs and 16 outputs would require tricky clock gating logic to save power.

5. Pipelined Repeater

A "pipelined repeater" is another name for a traditional QDI asynchronous half-buffer. It is implemented as c-elements followed by inverters for each data rail, with a NAND gate and inverter driving an inverted acknowledge backward. The pipelined repeater has only half a token of storage, since either its input or its output channel must be empty. The purpose of the pipelined repeater is to decouple the 4-phase handshake over long wires. Without the pipelined repeater, even if normal inverter repeaters were used, longer channels would operate at lower frequency. In $130nm$, a pipelined repeater is needed every $2mm$ to maintain $1.35GHz$, but the spacing can be increased for slower links.

5.1. Comparison to Synchronous Latches

The asynchronous pipelined repeater is analogous to a synchronous latch. Both designs can benefit from inverters automatically inserted on long wires. The comparison is interesting and quite relevant, since the power cost of cross-chip communication can be quite significant.

To compare a synchronous latch channel to an asynchronous channel, one must distinguish utilization and activity. Utilization is the fraction of peak bandwidth that is actually used, i.e. invalid synchronous padding doesn't count. Activity is taken to be the probability that an individual bit of data will change from one word to the next. Peak power consumption is at 100% utilization and 100% activity. The asynchronous system always has constant activity, since it has a return-to-zero phase to reset the channel between valid data.

To send a valid word over a synchronous channel requires 0 to 1 transitions per bit, depending on the activity. In addition, the clock input to the latch goes up and down once and drives all bits. The 1of4 asynchronous buffer takes 2 transitions to send 2 bits, plus raises and lowers an acknowledge once which gates all bits. So at rough estimate, the asynchronous and synchronous systems actually transition the same number of nodes for 100% activity.

The asynchronous system scales linearly with utilization all the way down to 0 power at 0% utilization. The synchronous system also scales linearly with utilization (assuming it holds the old data value for a padding cycle), but the clock load on the latches consumes a constant amount of power. The synchronous system also has a wide data-dependent power dissipation, since the power

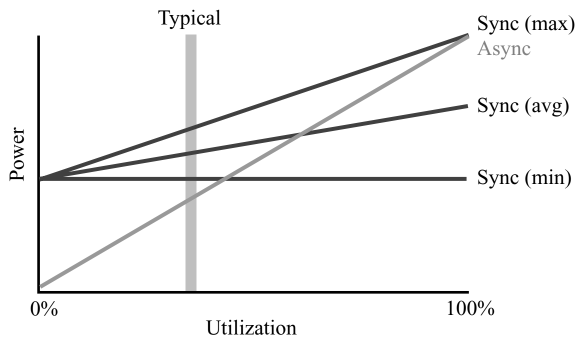

Figure 5. Power consumption of asynchronous pipelined repeaters versus synchronous latches.

also scales linearly with activity, unlike the asynchronous version. These characteristics can be summarized in Figure 5.

Note that two parts of this figure are of special interest. The first is peak power, that is 100% utilization and 100% activity. This is the maximum power consumption that needs to be considered for power distribution and heat dissipation requirements. Note that at rough estimate, the worst cast power for an asynchronous design is near that of the synchronous design. The other relevant part is the average power with a typical utilization and activity. This affects battery life and energy costs. In this domain, it is more likely that the asynchronous circuit will consume less power. Of course, the exact results vary with assumptions and implementations [6]. The point here is to explain the qualitative differences in power consumption of the two designs.

6. Clock Domain Converter

The clock domain converter consists of two independent circuits, a Synchronous-to-Asynchronous converter (S2A) for outbound data, and an Asynchronous-to-Synchronous converter (A2S) for inbound data. In both directions, full sender and receiver flow control is propagated across the boundary.

6.1. Synchronization Control Circuit

Both S2A and A2S use the exact same control circuit to decide when a transfer should occur. This control circuit accepts a valid signal from the input side and an enable from the output side, and must produce an enable to the input side and a valid signal to the output side. On the asynchronous side, these are interpreted as a 1of1 channel (data rail plus acknowledge). On the synchronous side, these are the request/grant signals. The converter control also receives the clock from the synchronous module and produces a "go" signal to latch data.

The control circuit must decide if a transfer should move forward on a certain clock edge. To do that, it arbitrates if both sides are "ready" on the rising edge of the clock against the case where they are not. This is done with a metastable circuit using cross-coupled NAND gates followed by a metastability filter. One half cycle is alloted for metastability resolution, and the result determines whether the next cycle advances or holds the old data.

6.2. Datapath

The S2A datapath uses the control output to latch the synchronous data into a flip-flop and perform the asynchronous handshake. It assumes that the asynchronous handshake will complete within a clock cycle without blocking. The A2S datapath is similar, and takes an asynchronous 1of4 code and latches it in flip-flops. It assumes that once the transfer is started, all asynchronous data is present and will complete the handshake within a clock cycle.

To guarantee that the asynchronous datapaths are ready, a completion detection circuit is needed to check that either all bits of an asynchronous input have arrived or that there is enough space in the asynchronous output channel. Instead of trying to complete all bits in a single c-element tree, we use a pipelined completion circuit which can sustain high frequency regardless of datapath width. The A2S completion sends a 1of1 channel to the control circuit to indicate that all data is present. The S2A completion sends a 1of1 channel to the control circuit to indicate that there is space in the output fifo. A few extra tokens are initialized on this channel to match the amount of storage available between the flip-flop and the completion detector. Figure 6 shows the structure of the S2A and A2S converters.

By checking that all input bits have arrived or that all output bits have buffer space, the converters can safely tolerate arbitrary relative skew between bits introduced by the asynchronous interconnect.

6.3. Latency

Latency is measured from when an input channel becomes valid to when the output channel becomes valid, assuming the link is empty to begin with. On the synchronous side, this is when the request is valid on a rising clock edge. The S2A has very little latency, since the asynchronous token is created directly from the rising clock through a small logic depth. On the A2S, there is a similar amount of asyn-

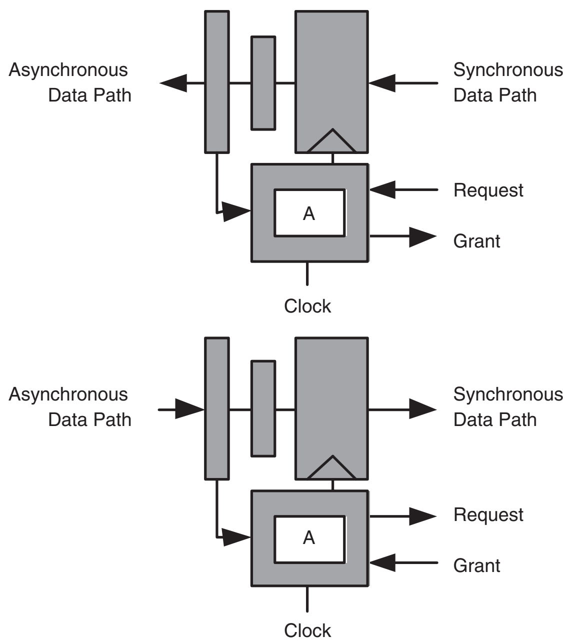

Figure 6. S2A and A2S clock domain converters.

chronous latency in completing the incoming asynchronous bits and making the control decision. However, the A2S also must wait for metastability resolution, and may be forced to wait an extra cycle if it "just misses" the sample window.

6.4. MTBF Analysis

In this design, there is a single potentially metastable synchronization associated with each transfer to or from a clock domain, regardless of datapath width. This must resolve within about $\frac{1}{2}$ of the clock cycle. To check the mean-time-between-failure (MTBF), we spiced the exponential decay of the metastable state across various process corners. This can predict resolution time as a function of input event spacing. We assume worst case simultaneous arrival time with a linearly distributed jitter of a few picoseconds, and compute the probability that the metastability will exceed the time allotted. From this we compute a MTBF for each transfer.

For our TSMC $130nm$ Nexus test chip, we find that even with all converters in the system encountering metastability on every cycle, we can achieve a MTBF of millions of years up to frequencies of $800MHz$. Due to the strong dependence of MTBF on frequency, we believe that for clock frequencies above $800MHz$, a longer metastability resolution time is preferable for high reliability systems. To do that, we have designed a new version of the converter which can be configured to allow either $\frac{1}{2}$ or 1 cycle for metastability resolution. Low frequency modules gain a latency advantage, while high frequency modules are still very safe.

6.5. Comparison to Standard Techniques

An S2A and A2S can be composed back-to-back to create a clock boundary bridge circuit. The asynchronous channels between them can be stretched and pipelined to span arbitrary on-chip distances at full frequency.

The standard technique for clock boundary crossing involves using a dual-ported register file to store the data (with writes and reads on the two clocks), and head/tail pointers in gray code transferred between the clock boundaries through metastable flip-flops. This is popular because it can be easily synthesized for a standard ASIC or FPGA. Although the datapath has no metastability, the pointer exchange actually has each bit metastable, although only one at a time can change, so the number of potentially metastable events is still only 2 per transfer.

Our converter is much simpler, with only a single metastable node and simultaneous computation of forward validity and backward acknowledges. The datapath is essentially a synchronous flip-flop combined with a small amount of asynchronous buffering, which is smaller than a dual-ported register file. The latency of our design is also much less. Others have developed similar optimized designs [1, 7], but the standard technique is still the most widespread.

7. Verification and Test

The Nexus interconnect will soon appear in a number of commercial chips, including those developed by Fulcrum Microsystems as well as our partners. Several challenges had to be overcome in order to use asynchronous technology in a commercial context. The most interesting of these are discussed briefly.

7.1. Noise Analysis

Asynchronous circuits are susceptible to noise glitches for a larger portion of their cycle than a synchronous design with flip-flops and combinational logic. Although our digital design never glitches, analog noise from charge-sharing or capacitive-coupling can cause glitches. Dynamic logic susceptible to charge sharing is always contained within a small cell and can be checked locally. Various design guidelines and netlist transformations are used to fix any charge-sharing problems without resorting to pre-charging internal nodes. Capacitive-coupling tends to affect longer wires, but

these are strongly driven by inverters and also have inverters or pipelined repeaters spaced at reasonable distances. Also, the frequent use of 1of4 encodings means that usually only 1 of a group of 4 wires could be an aggressor.

Both forms of noise were exhaustively spiced using an in-house tool for setting up and simulating the relevant portions of large circuits. All circuits are verified to perform correctly with a worst-case combination of simultaneous capacitive-coupling aggressors and charge-sharing.

7.2. Timing Analysis

The QDI nature of the interconnect relaxes much of the timing analysis problem, since timing variance may affect performance but won't result in failure. Transistor sizes were selected automatically by a sizing tool which uses floorplanning information to estimate RC parasitics. After layout, performance of the core blocks is verified by spice simulation over process, voltage, and temperature corners. On the wires between blocks, a target delay budget must be satisfied, or more pipelined repeaters are inserted. On the synchronous converters, however, normal set-up and hold times apply and may be checked with static timing tools.

7.3. Fault and Delay Testing

The extensive use of domino and dynamic logic presents challenges for fault modeling, although these are largely similar to full-custom synchronous designs which also use domino logic. We use an equivalent model for our dynamic gates and a commercial tool to fault grade our circuits. We have not yet adapted a commercial ATPG tool to generate test patterns automatically, but given the simple datapath of Nexus we can manually generate patterns.

Fault testing requires some additional circuitry. In each converter, we include a loopback mux on the synchronous side, which can "bounce" an incoming burst back into the crossbar after swapping its FROM with some data bits in the first word to create the TO and a modified first word. One of the Nexus ports has an asynchronous BIST assist module which launches bursts into the interconnect and bounces them through 2 ports and then back, and then around again for a configurable number of iterations. This can test all links between modules with various patterns and even at full speed. The final burst data is checked on the synchronous side with a standard scan chain. Many faults result in deadlock, which are easily observed by the lack of valid data at the expected time. Despite the asynchronous nature of the interconnect, the test stimulus and response can be made completely deterministic merely by waiting a "suitable" time before looking at the results. This can also be used to detect speed faults or perform speed binning.

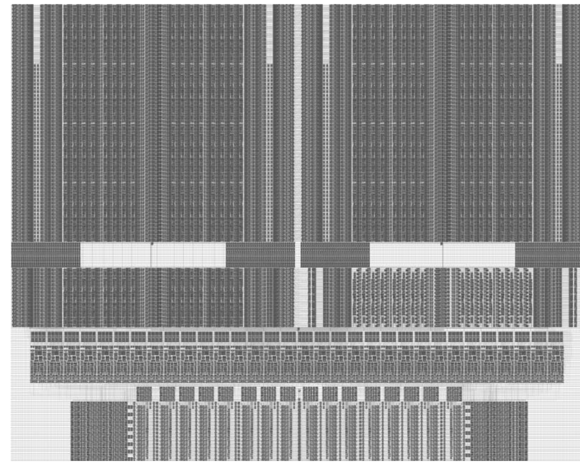

Figure 7. Layout of 16 port 36 bit crossbar – from bottom to top, the input/output control, repeat-until-tail, 4-bit and 1-bit crossbars, then the remaining 32-bits of crossbars.

8. Characterization Results

The Nexus Interconnect has been fabricated and characterized in numerous processes with minor design variations. It has been functionally correct and high performance in all of them. In TSMC $180nm$ G, all Nexus components operate up to $450MHz$ at $1.8V$ and $25C$. In TSMC $150nm$ G, they operate at $480MHz$ at $1.5V$ and $25C$. Most recently, Nexus has been fabricated in TSMC $130nm$ LV, with both Low-K and FSG insulator. These results will be presented in more detail.

In $130nm$, the 16 port 36 bit crossbar itself is $1.75mm^2$. Each pipelined repeater is $0.025mm^2$ for both directions with 36 bit data, 1 bit tail, and 4 bits control. The clock domain converters are $0.2mm^2$ in this design, but will soon be optimized to $0.1mm^2$ (they were mostly automated layout). A typical Nexus system with all 16 ports used and an average of 2 pipelined repeaters per link would be $4.15mm^2$ total area. This is a small fraction of the typical die area of the chip it would be used in. Figure 7 shows the layout of the crossbar.

Correct operation was verified from $-55C$ to $125C$ and $0.7V$ to $1.4V$. Figure 8 summarizes the frequency over voltage at $25C$ for both the Low-K and FSG processes. Energy measurements were also made to characterize the energy per bit transferred, in pJ/bit. This was measured for a 2-word burst from a synchronous module to itself, and doesn't depend on the data values transferred. Key frequency, la-

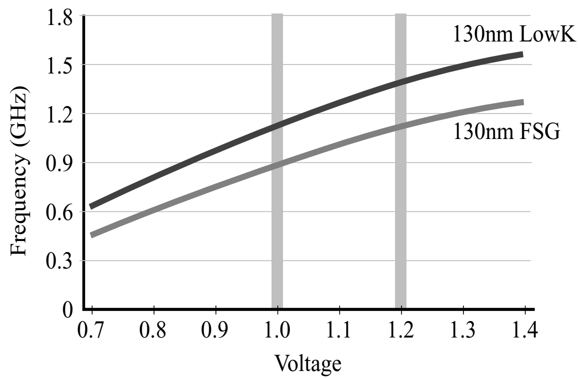

Figure 8. Nexus frequency versus voltage.

tency[2], and energy results at $25C$ are summarized as follows:

Process	V	GHz	ns	pJ/bit
Low-K	1.2	1.35	2.0	10.4
Low-K	1.0	1.11	2.4	7.0
FSG	1.2	1.10	2.5	11.2
FSG	1.0	0.87	3.1	7.6

9. Conclusions

We have presented the design and characterization of Nexus, a globally asynchronous, locally synchronous (GALS) interconnect system with favorable speed and power characteristics. The asynchronous parts of the design follow the QDI timing model and the Integrated Pipelining design style. The components of the system include a 16 port, 36 bit asynchronous crossbar with arbitration and flow control, pipelined repeaters to communicate over long wires, and clock domain converters to connect to synchronous modules. Nexus achieves $1.35GHz$ frequency and $780Gb/s$ cross-section bandwidth in a $130nm$ process.

As shrinking process geometries and higher frequencies increase the difficulty of global clock distribution, and as timing variability increases, we believe that a GALS approach is the best way to design large System-on-Chip's. The Nexus interconnect is a complete and proven solution to this problem. As other asynchronous components are introduced, Nexus will make the gradual migration from synchronous to asynchronous cores easier. Soon, an SoC designer will be free to choose whichever design style provides the best solution for each component.

[2] Reported latency does not include the $\frac{1}{2}$ to $\frac{3}{2}$ clock period of the receiving module. Latency is difficult to measure directly, and is inferred from a combination of lab measurements and spice simulation.

References

[1] S. Moore, G. Taylor, R. Mullins, P. Robinson. "Point to Point GALS Interconnect," *Proceedings of the 8th International Symposium on Asynchronous Circuits and Systems.* IEEE Computer Society Press, 2002.

[2] W.J. Bainbridge, S.B. Furber. "Delay Insensitive System-on-Chip Interconnect using 1-of-4 Data Encoding," *Proceedings Async 2001.* IEEE Computer Society Press March, 2001.

[3] A.J. Martin. "The Limitations to Delay-Insensitivity in Asynchronous Circuits." *Sixth MIT Conference on Advanced Research in VLSI,* MIT Press, 1990.

[4] A. Lines. "Pipelined asynchronous circuits." M.S. Thesis, Caltech CS-TR-95-21, 1995.

[5] A.J. Martin, A. Lines, R. Manohar, M. Nyström, P. Penzes, R. Southworth, U. Cummings, and T.K. Lee. "The Design of an Asynchronous MIPS R3000 Processor," *Proceedings of the 17th Conference on Advanced Research in VLSI.* IEEE Computer Society Press, 1997.

[6] K. Stevens. "Energy and Performance Models for Clocked and Asynchronous Communication," *Proceedings of the 9th International Symposium on Asynchronous Circuits and Systems.* IEEE Computer Society Press, 2003.

[7] A. Chakraborty, M. Greenstreet. "Efficient Self-Timed Interfaces for Crossing Clock Domains," *Proceedings of the 9th International Symposium on Asynchronous Circuits and Systems.* IEEE Computer Society Press, 2003.