# A 50 Gb/s 32 × 32 CMOS Crossbar Chip using Asymmetric Serial Links*

Kun-Yung Ken Chang, Shang-Tse Chuang, Nick McKeown, and Mark Horowitz
Computer System Laboratory, Stanford University
Stanford, CA 94305

## ABSTRACT

A 32 × 32 synchronous crossbar chip was designed in a 0.27μm CMOS technology for use in a high-speed network switch [1]. The crossbar chip uses 32 *Asymmetric Serial Links* [2][3] to achieve high speed at the interfaces and to reduce both power and area. The crossbar switch core is implemented with static CMOS multi-stage multiplexors with multicast capability. The chip operates successfully with links running at 1.6 Gb/s. The measured bit-error-rate is < $10^{-14}$ when all channels and the switch core are operating. The crossbar chip consumes 5W and provides a total bandwidth above 50 Gb/s.

## Architecture

Crossbar switches are increasingly used for high capacity network switches [1]. Using high-speed serial links for the I/O interface of a crossbar switch chip can reduce the number of I/O pins required to provide a given bandwidth. A crossbar chip, which has many links converging on it, tends to be large and power-hungry because of the silicon area and power consumed by the timing circuits (usually PLLs) and transceivers. To reduce both power and area, we used *Asymmetric Serial Links*, which adjust the transmitter and receiver clocks on the port chips (the *Smart End* of the link) while leaving the clocks on the crossbar chip (the *Dumb End*) fixed. Each crossbar chip interfaces with as many as 32 network port chips in our design.

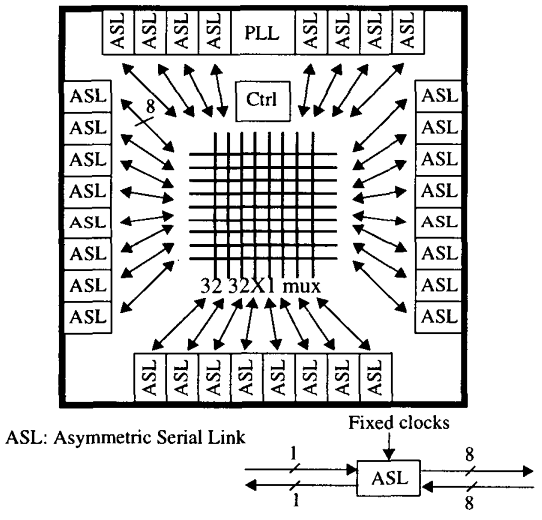


**Figure 1**: Chip architecture of the 32 × 32 crossbar

Figure 1 shows the physical architecture of the 32 × 32 crossbar chip, which consists of 32 *Asymmetric Serial Links*, a PLL, an 800MHz clock distribution tree (not shown), a 32 × 32 switch core, and a crossbar controller. The speed of the pin interface is 1.6 Gb/s. The switch core and the majority of the digital logic on the chip runs at 200MHz. The speed conversion between these two clock domains is handled by serial-to-parallel and parallel-to-serial converters inside the *Asymmetric Serial Links*. 200MHz is 1/8 of the bit rate of the serial links and hence all the datapaths on the chip are 8 bits wide, including the switch core and the interface registers between the serial links and the switch core.

The PLL serves two purpose. First is for clock multiplication. Second is to compensate for delay variations in the clock distribution.

During normal operations, all 32 serial link blocks work as *Dumb Ends* of the *Asymmetric Serial Links*, which do not adjust their own transmitter and receiver clocks. However, for testing purposes, two serial link blocks were designed as *Smart Ends* with PLLs to adjust their clocks, but may also be operated as *Dumb Ends*.

The crossbar controller manages the initial calibration process, disabling links before they are properly synchronized. As a result, when a serial link is disconnected, it will never be synchronized, so it will always be disabled by the controller without any explicit logic.

The *Asymmetric Serial Links* use differential-pair open drain transmitters and current-integrating receivers [6]. The data are sent differentially on both edges of the clocks on the links. PMOS pull-up transistors are used as the termination resistors at each receiver. Dual-loop PLLs with digital-controlled phase interpolators [5] are used. The phase interpolators are controlled by digital phase control logic inside the *Smart End.*

## Data Transmission Protocol

The scheduling of the crossbar chip is performed by a separate scheduler chip in the system, which sends the routing tag to the crossbar chip through the port chips interfacing with the crossbar. The routing tag is embedded inside data frames.

| Hdr | Data7 | Data6 | Data5 | Data4 | Data3 | Data2 | Data1 | Data0 |
|---|---|---|---|---|---|---|---|---|
| 71 | | | | | | | | 0 |

Bit 71: idle bit — Bit 65, 64: reserved bits for system use
Bit 70-66: reverse routing tag — Bit 63-0: 8 bytes data

**Figure 2**: Data format in the crossbar switch

Figure 2 shows the data format used by the crossbar chip. The data are grouped into 9-byte frames. The first byte is the header and the other eight bytes are the payload. In the header byte, the first bit is called the idle bit, which indicates whether the frame contains useful data. When the idle bit is set, the frame is used for calibrating the serial links. The next 5 bits in the header are the reverse routing tag [1], which indicates the port number of the input port (not the output port) that is going to send data to this port in the same frame time. The remaining two bits of the header are for system use and are not used by the crossbar chip.

The use of reverse routing tags makes each port able to send data to multiple output ports when those ports have identical tags. As a consequence, multicasting is supported by the switch core.

The framing of all 32 ports is synchronized in the system. Hence, the switch core is a synchronous crossbar switch that is configured at once for all 32 ports.

## Synchronization

The *Asymmetric Serial Links* work in pairs, and are also synchronized in pairs. In other words, the *Smart-to-Dumb Link* and the *Dumb-to-Smart Link* are synchronized together.

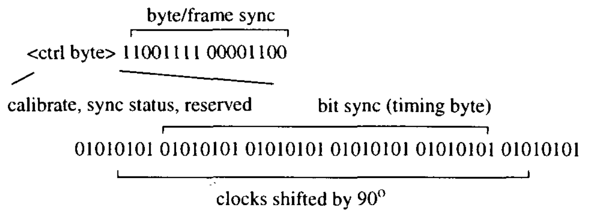


**Figure 3**: Calibration Sequence

The synchronization of the crossbar chip includes bit, byte and

* Supported by Texas Instruments and Cisco Systems

frame synchronization. A nine-byte calibration frame, shown in Figure 3, is used for synchronization. The first byte is called the control byte, which consists of one idle bit, four synchronization status bits, and three reserved bits for system use. The four synchronization status bits consist of clock early, clock late, byte sync, and frame sync of the other link. The second and third bytes, which contain consecutive 1 or 0 pairs, are for byte and frame synchronization. The remaining six bytes, containing alternating 1s and 0s, are for bit synchronization.

Figure 4 shows a pair of *Asymmetric Serial Links.* At the beginning of chip bootup, the calibration packet is sent over the dumb-to-smart link and a 11000001 pattern is sent over the smart-to-dumb link continuously. The latter is to avoid any false locking in the smart-to-dumb link. The dumb-to-smart link, which adjusts its receiver clock at the *Smart End*, is thus synchronized first (at the byte/frame level and then at the bit level). Once the dumb-to-smart link is synchronized, the smart transmitter starts sending calibration packets. The synchronization status of the smart-to-dumb link is sent back to the *Smart End* through the dumb-to-smart link, which is reliably synchronized at this point.

After each calibration frame, a majority vote is taken on the early and late bits received from the fifth to eighth bytes to determine whether the clock is early or late. In the dumb receiver, this information is sent back to the smart transmitter using the clock early and late bits of the control byte mentioned earlier.

The synchronization status in the control byte notifies a serial link whether the other link in the pair is synchronized or not. When the links of both directions are synchronized, the serial link pair is ready for use.

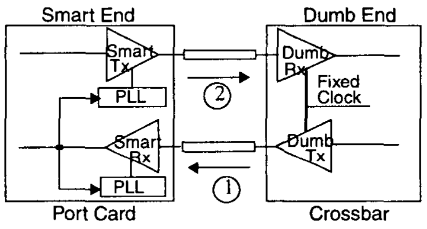


**Figure 4**: Asymmetric Serial Link Synchronization

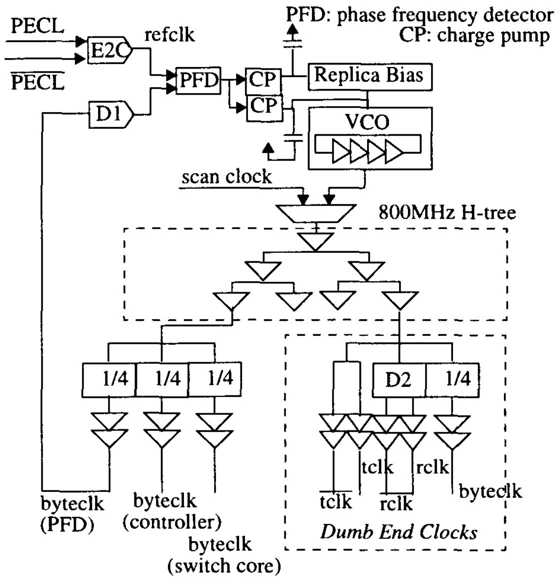
To increase the timing margin of the *Asymmetric Serial Links*, the timing information for adjusting the clocks is obtained by shifting the data receiver clock by 90°. This means that the same receiver and clock buffer chain are used to receive both data and timing information. As a result, the only systematic offset for the 90° clock shifting comes from the layout mismatch inside the VCO, which can be minimized by careful layout effort.

## PLL and Clock Distribution

Figure 5 shows the clock generation on the crossbar chip, which is the *Dumb End* of the *Asymmetric Serial Links*. The PLL uses a 4-stage VCO with symmetric loads and replica feedback biasing [4] to reduce its supply sensitivity. The input reference clock is a 200 MHz PECL clock, which is amplified by an PECL-to-CMOS converter to full swing for the phase frequency detector. A delay matching circuit is inserted in the PLL feedback loop to compensate for any delay variation in the converter, which has 450ps delay in the typical process corner.

The 800MHz VCO output is multiplexed with the scan clock of the scan chain on the chip before driving the balanced clock distribution tree. By multiplexing these two clocks, the scan chains on the chip do not have problems with hold-time violation because the scan clock is distributed in a balanced fashion by the 800MHz clock distribution tree.

Each tail of the 800MHz clock tree drives local clock generators for the serial link blocks, the switch core, and the crossbar controller. The local clock drivers for the serial links generate byte clocks (200MHz), transmitter and receiver clocks (800MHz). The delay matching circuit in the receiver clock path matches the delay of the clock-to-Q of the divide-by-4 counter and the transmitter output. The clock buffers for all three clocks are designed with approximately the same delay. Other local clock drivers, such as the ones for the switch core and the crossbar controller, only generate the byte clocks, which have the same delay in their clock buffers as the *Dumb Ends*. One of these byte clocks is fed back to the PLL. As a result, the timing of transmitter output, receiver sampling time, and all the byte clocks are phase-locked to the external reference clock.



D2: delay matching circuit of the clock-to-Q of the 1/4 counters

**Figure 5**: Clock Generation

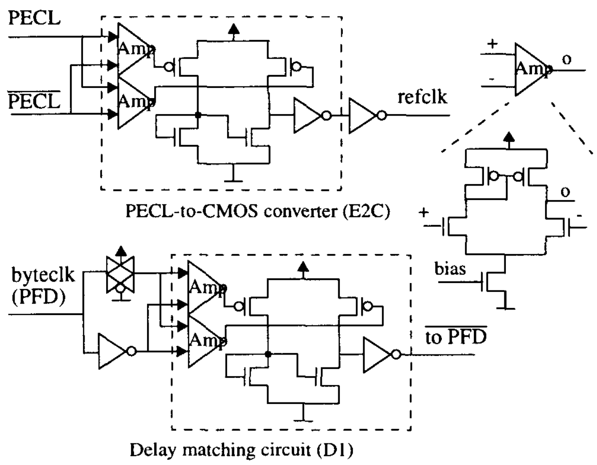

PECL
$\overline{\text{PECL}}$
Amp
Amp
refclk
PECL-to-CMOS converter (E2C)
+
-
Amp
o
o
-
+
bias
byteclk (PFD)
Amp
Amp
$\overline{\text{to PFD}}$
Delay matching circuit (D1)

**Figure 6**: PECL-to-CMOS converter and its delay matching circuit

Figure 6 shows the PECL-to-CMOS converter and the delay matching circuit shown in Figure 5. The converter includes a differential-to-single-ended converter [4], followed by two inverters. The first inverter is skewed in size to generate 50% duty-cycle of the full-swing clock. The second inverter is to drive the signal to the input of the phase frequency detector. The delay matching circuit takes the feedback full-swing byte clock from the clock distribution tree to drive a pass gate and an inverter to generate complementary clock signals. These clocks then drive a differential-to-single-ended converter that has the same design as in the PECL-to-CMOS converter. The inverter and pass gate are sized small to slow down the differential-to-single-ended converter so that the total delay of the two circuits are almost matched. Simulations show that the maximum mismatch is 44ps across 400mV variations on the input common mode with the same supply voltage.

Figure 7 shows the physical implementation of the 800MHz clock tree. To provide the minimum skew, we used an H-tree topology, ensuring that the arrival time of clocks to each tail point of the clock tree is approximately the same. The clock skew of the 800MHz clock tree affects the timing margin for synchronizing the divide-by-4 counters. The worst case timing margin comes from the edge of each clock tree. To satisfy the setup timing between the two divide-by-4 counters at the adjacent tails, the clock skew must be smaller than a bit time (625 ps).

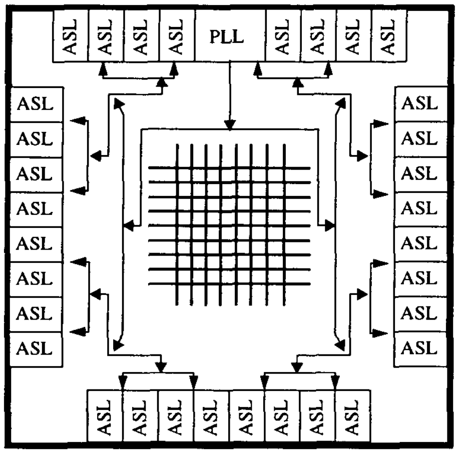


**Figure 7**: 800MHz H-tree Clock Distribution

## Switch Core

The switch core, which runs at the byte clock rate, is implemented with 32 static CMOS $32 \times 1$ byte-wide multiplexors. Figure 8 shows the datapath of the switch core. The 8-bit data are first registered at the output of each port before being sent to the switch core. Then the data are driven to the input lines and the tag decoder of the same port. If the data is a header byte, the tag decoder changes its setting and reconfigures the multiplexors. The data bytes are multiplexed and registered at the output of the switch core. The registers are physically located at the end of the output lines. As a result, the latency of the switch core is one byte-clock cycle.

With reverse routing tags, the tag from input port i configures the multiplexors of output port i. Since each input line goes to the multiplexors of all the output ports, when multiple multiplexors for different output ports are turned on for the same input line, the data bytes are multicast to all of those ports.

Each 32-to-1 multiplexor is implemented by a series of three multiplexors: 2-to-1, 4-to-1, and 4-to-1. To minimize the wiring for both input and output, both sets of wires run straight through the entire switch core, with input wires running horizontally and output wires vertically, as also shown in Figure 8. The 2-to-1 multiplexors run between two adjacent 8-bit blocks to select between two input ports. The first 4-to-1 multiplexors are placed across 8 8-bit blocks and the second 4-to-1 multiplexors are distributed across all 32 blocks in the vertical direction. Each of the output wires is 1.6mm long. The extreme length of these wires requires increasing the size of the multiplexors to reduce the large signal rise/fall time. However, larger multiplexors also have larger output loading. The problem is magnified because the "on" tri-state buffer has to drive a long wire and the output loading of all other "off" tri-state buffers. Serializing the multiplexors relieves the output self-loading problem by reducing the total number of tri-state multiplexors (to 4) on the same output wire.

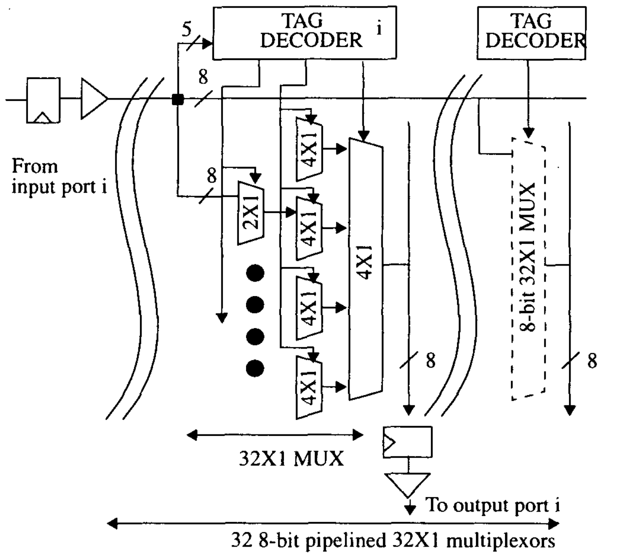


**Figure 8**: Switch core architecture and reverse routing tag

## Measurement Setup and Results

A $32 \times 32$ crossbar test chip was designed and fabricated using Texas Instruments' 0.27μm CMOS technology. The *Smart Ends* are used as the port chips to test the *Dumb End* crossbar chip.

Each test channel for the differential links consists of a pair of 6-inch FR4 traces, 10-inch coaxial cables, and BGA package traces, all with 50-ohm impedance. The SMAs for connecting the boards and cables were measured to have 25-ohm impedance and about 60ps delay.

Two different types of tests were run to validate both the functionality and speed of the crossbar chip. First, four serial links, including both *Smart Ends*, are implemented with 7-bit PRBS (Pseudo-Random-Bit-Sequence) generators and verifiers. The PRBS tests a single serial link without interacting with the switch core on the crossbar chip, while other links and the switch core simply generate noise to the chip supply and substrate.

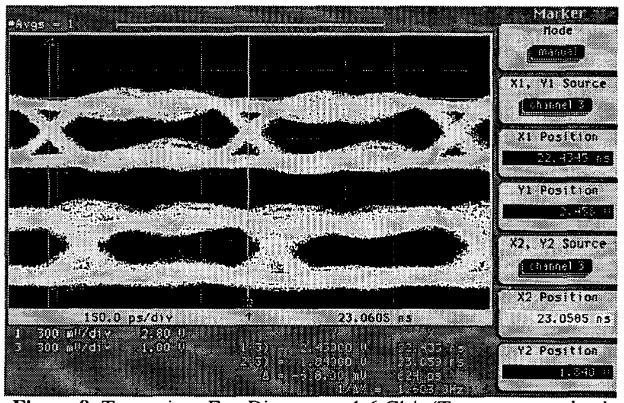


**Figure 9**: Transmitter Eye Diagram at 1.6 Gb/s (Top: *smart-to-dumb* link, Bottom: *dumb-to-smart* link)

Figure 9 shows the eye diagram of the transmitter output at 1.6 Gb/s from an error-free single-link PRBS test running over 24 hours, demonstrating a bit-error-rate below $10^{-14}$. The reflections shown in the eye diagram are due to the capacitive SMA connectors. The jitter of the *dumb-to-smart* link is larger because its clock comes from the 800MHz clock distribution tree on the crossbar chip, while the clock of the *smart-to-dumb* link is self-generated inside the *Smart End* and thus has shorter buffer chains.

Figure 10 shows the timing margin of the two links at the receiver output. This was measured by the phase error injection method [2]. The phase error injection method measures the timing margin of a link by injecting phase errors on the transmitter/receiver data clock for the smart transmitter/smart receiver link during the PRBS test. When a PRBS error is detected, the amount of the injected phase error on both early and late directions is at the timing margin of the link. The histograms of the centering of the receiver sampling points after 20 phase error injections are in the middle of the diagrams. The two sides of each diagram show the failing points for the 20 measurements. The white area represents the timing margin of the links and the gray area represents the time when the links fail. Figure 10 shows that the receiver sampling points are almost at the center of the data eye. This indicates that the 90° phase shifting of the data receiver clock for timing recovery compensates for any systematic offsets effectively.

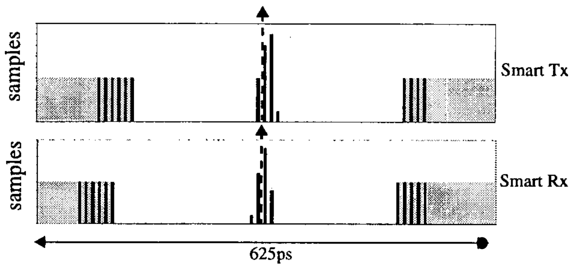


**Figure 10**: Link timing margin at 1.6Gb/s

To test the switch core with all of the serial links transmitting data at full speed, we configured several crossbar chips as *Smart Ends*. Each smart end is assigned a port number. The transmitted data are associated with the port number. During the test, each port uses another 7-bit PRBS generator to generate pseudo-random routing tags and sends them along with the encoded data. Since each tag carries the data source for the port, by extracting the port number from the incoming data, one can identify whether the data are sent correctly through the switch core and the serial links. For such tests, our experiments showed that each link can run reliably at a speed of 1.6 Gb/s, with the switch core running at a 200MHz clock rate. As a result, the crossbar chip can provide a raw data bandwidth above 50 Gb/s with all of the 32 channels running.

Furthermore, our experiment shows that the serial links can run as fast as 1.92 Gb/s without interfacing with the crossbar core. Figure 11 shows the timing margin of the serial links measured at 1.92 Gb/s. The offset from the center is caused by speed limitations in the digital logic. Even so, the timing margin is still large and close to the ideal center. This indicates that either the digital logic, including the crossbar core, or the skew on the clock distribution limits the speed of this crossbar chip. Therefore, with more careful layout effort on the digital logic, the switch core, and the clock distribution tree, we believe that the crossbar chip could provide a raw data bandwidth of over 60 Gb/s with each serial link operating at 1.92 Gb/s.

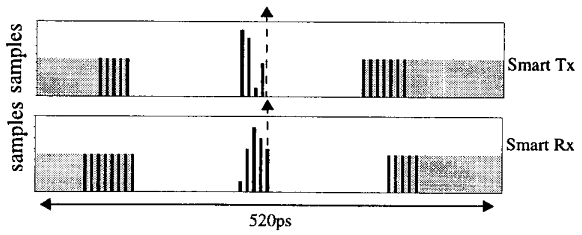


**Figure 11**: Link timing margin at 1.92 Gb/s

Figure 12 shows the chip microphotograph of the $32 \times 32$ crossbar chip. There are 30 *Dumb* serial link blocks and 2 *Smart* serial link blocks. The chip has 300 pins, including signal, power, and ground pins. Table 1 summarizes the specification of the crossbar chip.

## Acknowledgments

We thank Bill Ellersick, Ken Yang, Dave Nakahira, and Lance Hammond from Stanford and Martin Izzard, Ah-Lyan Yee, and Rhys Venett from Texas Instruments for their support and assistance.

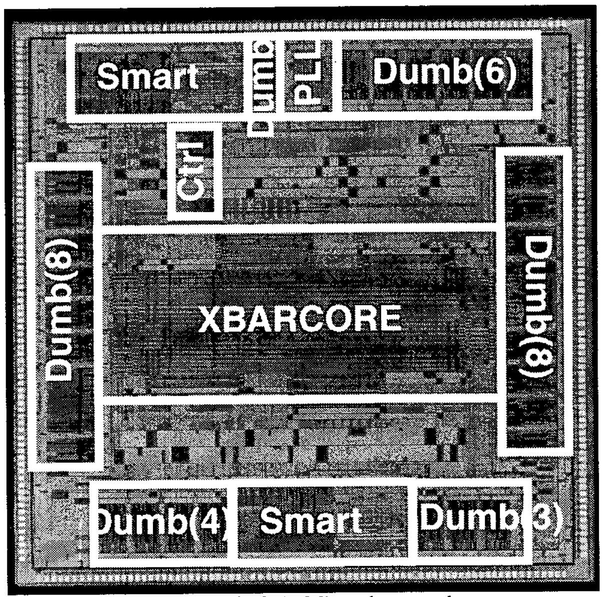


**Figure 12**: Chip Microphotograph

| | |
|---|---|
| **Supply Voltage** | **2.5V** |
| **Technology** | **TI 0.27μm CMOS** |
| **Die Size** | **8.5mm x 8.5mm** |
| **Package** | **352-pin PBGA** |
| **Transistor Count** | **0.5 million** |
| **PLL Range (VCO rate)** | **200MHz - 1.44GHz** |
| **Power Dissipation @ 1.6 Gb/s/link** | **5W (full crossbar)** |
| **Jitter (@800MHz)** | **75ps p-p and 12ps rms** |
| **Supply Sensitivity (pk-pk)** | **1.4ps/mV (@800MHz)** |
| **Max Data Rate/Serial Link w/o crossbar core** | **1.92Gb/s** |
| **Max Data Rate/Serial Link with crossbar core** | **1.6Gb/s** |
| **Max Crossbar Bandwidth** | **51.2Gb/s** |

**Table 1: Chip Summary**

## References

[1] N. McKeown, et al. "Tiny Tera: A Packet Switch Core," IEEE Micro, Jan/Feb. 1997.
[2]K. Chang, et al. "A 2Gb/s/pin CMOS Asymmetric Serial Link," VLSI Circuit Symposium, 1998.
[3]K. Chang, et al. "A 2Gb/s Asymmetric Serial Link for High-Bandwidth Packet Switches," Hot Interconnects '97, pp171-179.
[4] J. Maneatis, "Low-Jitter Process-Independent DLL and PLL Based on Self-Biased Techniques," IEEE JSSCC, Nov. 1996.
[5] S. Sidiropoulos, M. Horowitz, "A Semi-Digital Dual Delay-Locked Loop," IEEE JSSCC, Nov. 1997.
[6] S. Sidiropoulos, M. Horowitz, "A 700 Mbps/pin CMOS Signalling Interface Using Current Integrating Receivers," IEEE JSSCC, May 1997.