Embedded Processor Cores (Hard and Soft)

Electronic design can be realized in hardware (logic gates/registers) or software (instructions executed on a microprocessor).

The trade-off is determined by how fast it needs to run.

- picoseconds and nanosecond range (hardware only)
- microsecond range (either, where most of the options are)
- millisecond range (mainly software)

  This is where interfaces reside, e.g., for reading switch positions and flashing LEDs.

  It's a pain to slow hardware down for these functions, using large counters.

The facts are that most designs make use of microprocessors in some form.

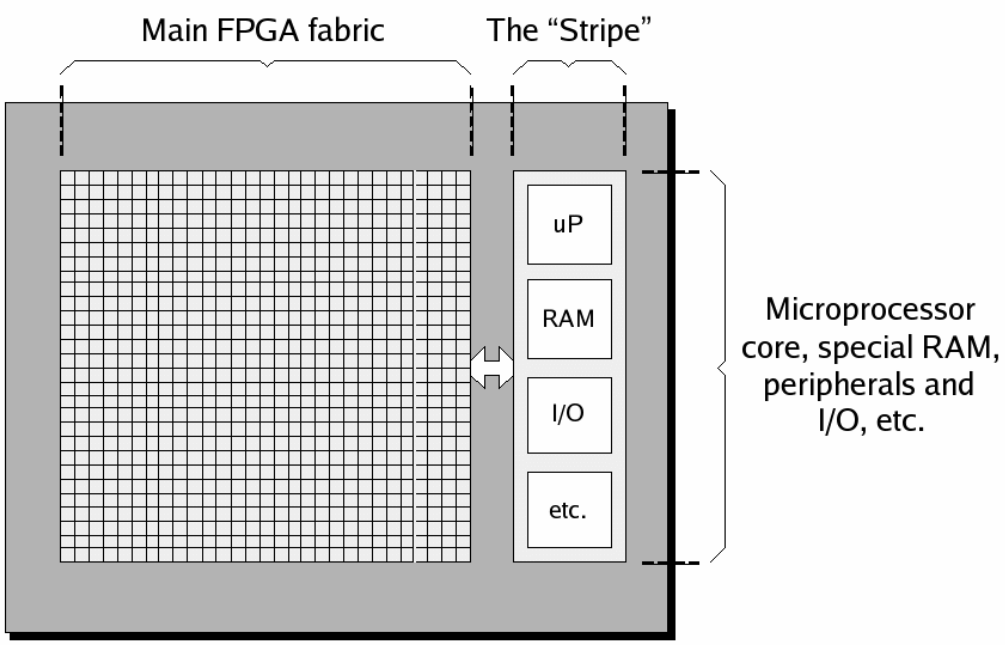
Until recently, these appeared as discrete chips on the PCB -- now they appear as embedded **microprocessor cores** within the FPGA.

Embedded Processor Cores (Hard and Soft)

A **hard** microprocessor core is implemented as a dedicated, predefined block.

There are two options for its integration.

- In a strip (called the Stripe) to the side of the FPGA fabric.



The Design Warrior's Guide to FPGAs,
ISBN 0750676043,
Copyright(C) 2004 Mentor Graphics Corp

In this case, all components are integrated onto a single piece of silicon or fabricated onto two chips and packaged in a *multichip module* (MCM).

Embedded Processor Cores (Hard and Soft)

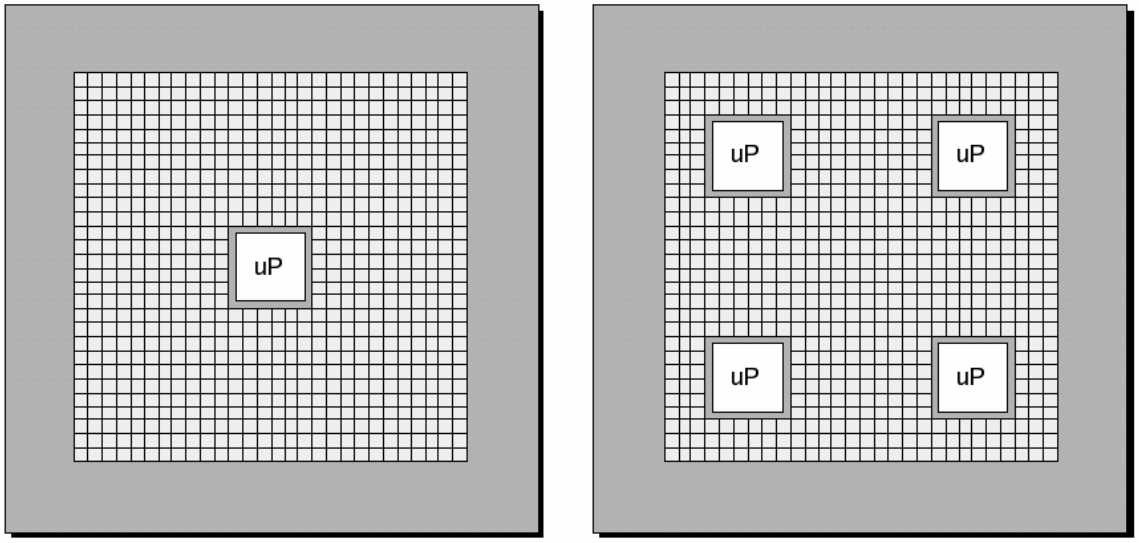
- In a strip (called the Stripe) to the side of the FPGA fabric (cont).

  Advantages:

  The main FPGA fabric is identical for chips with and without the microprocessor.

  The FPGA vendor can bundle other features in the *stripe*, such as memory, special peripherals, etc.

- Embed directly into the FPGA fabric



(a) One embedded core (b) Four embedded cores

Embedded Processor Cores (Hard and Soft)

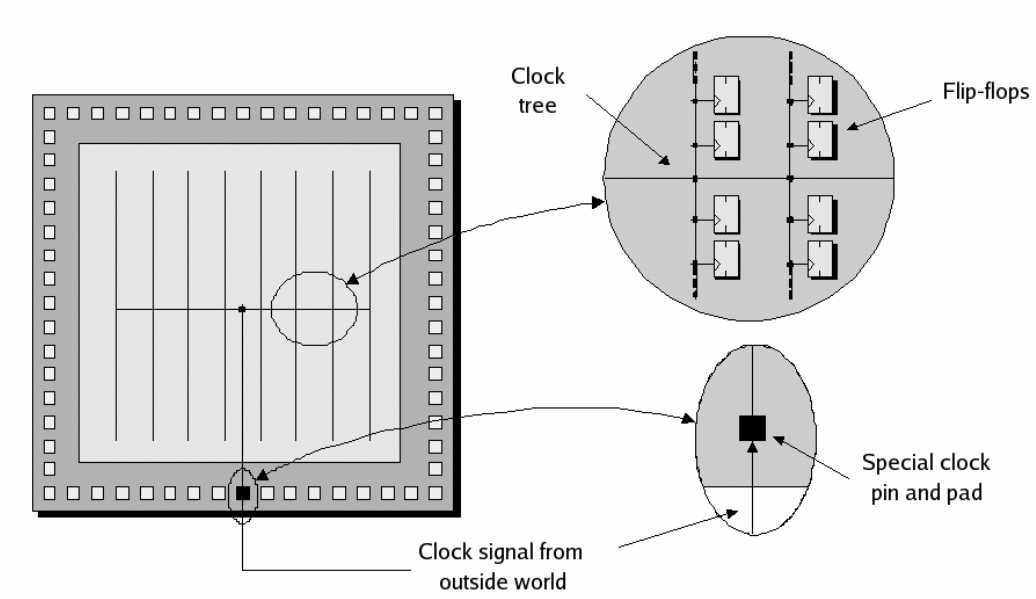
For *soft microprocessor cores*, a group of PLBs are configured as the micro.

Soft cores are slower than their hard-core counterparts (30-50% slower).

The advantage here is that you don't add one unless you need it, and you can add as many as you like, until resources run out.

Clock Trees and Clock Managers

All synchronous FFs in the chip are driven by an external clock signal.



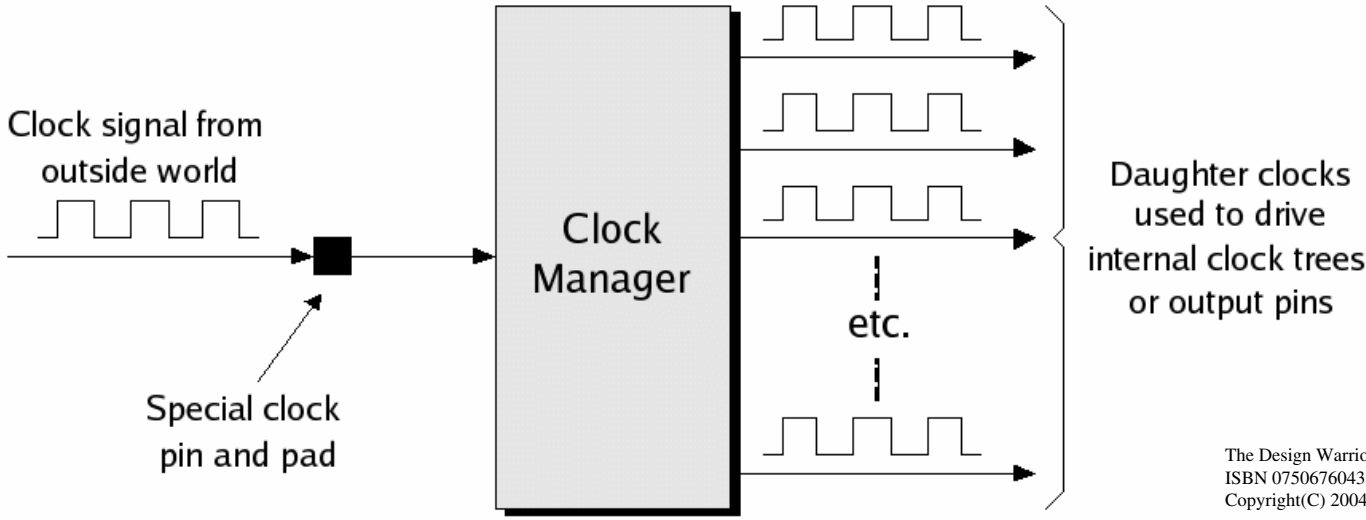
The Design Warrior's Guide to FPGAs,
ISBN 0750676043,
Copyright(C) 2004 Mentor Graphics Corp

Clock Trees and Clock Managers

The clock tree is routed using special tracks in the FPGA, and is designed to minimize **clock skew**.

In reality, there are multiple clock pins to support *multiple clock domains* within the FPGA.

The *clock manager* is a special hard-wired function that can receive the external clock signal and generate **daughter clocks**.



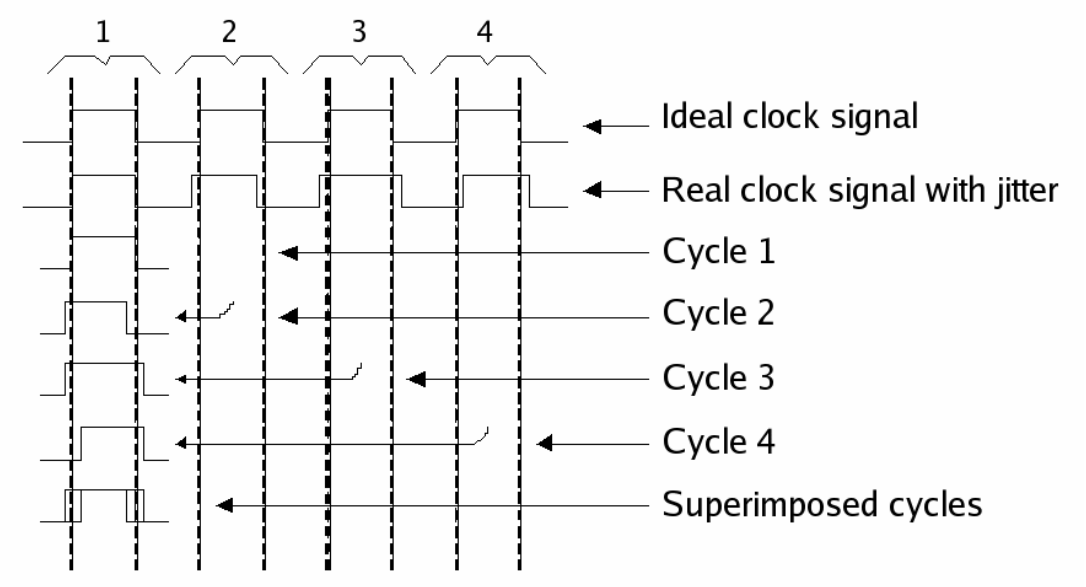
The Design Warrior's Guide to FPGAs,
ISBN 0750676043,
Copyright(C) 2004 Mentor Graphics Corp

Clock Trees and Clock Managers

The daughter clocks can then be used to drive internal clock trees, or output pins for distribution to other chips on the PCB.

Clock managers can support the following features:

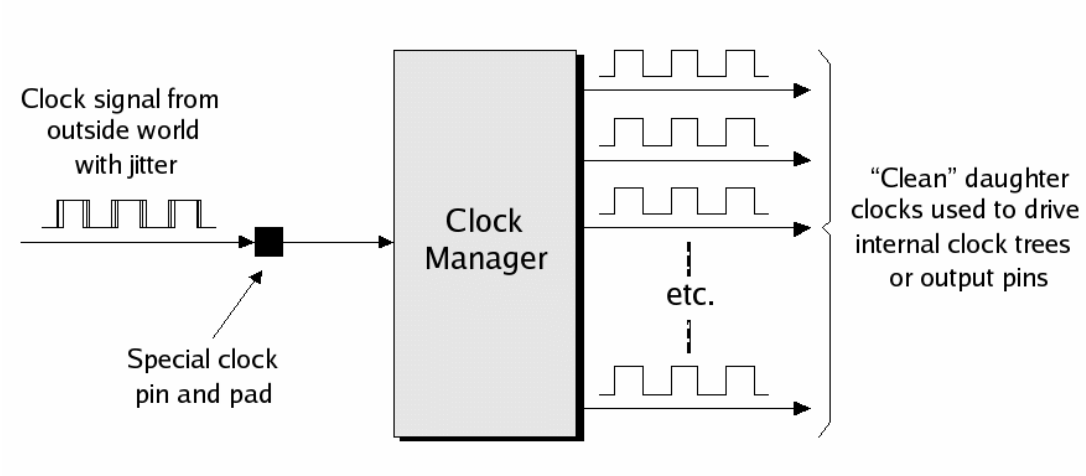
- *Jitter removal*: Uncertainty in the exact arrival time of each clock edge results in jitter.



The clock manager can be used to *detect* and *correct* for this jitter, and therefore provide **clean** daughter clock signals.

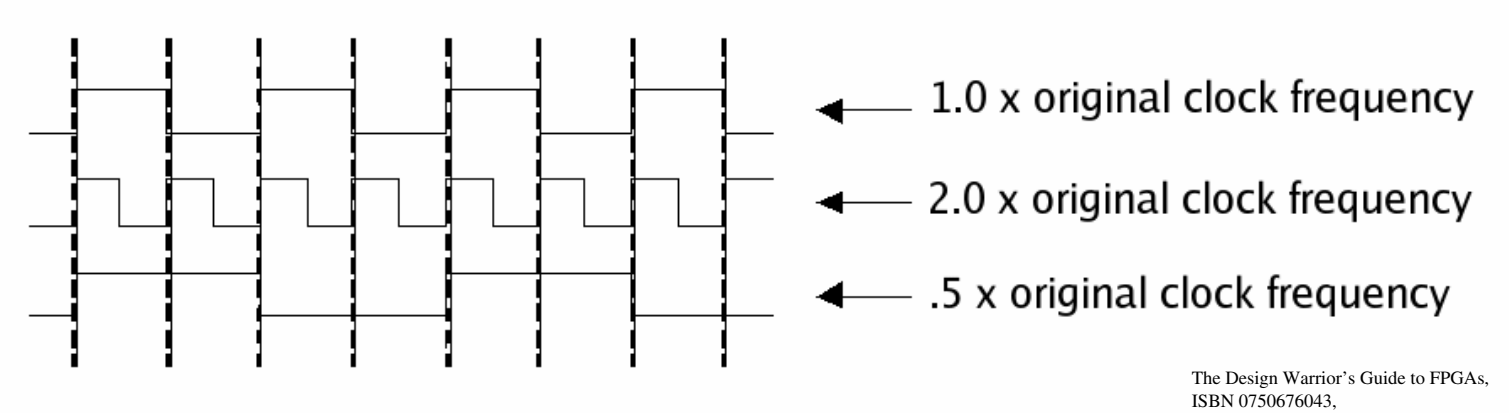
Clock Trees and Clock Managers

- *Jitter removal*



The Design Warrior's Guide to FPGAs,
ISBN 0750676043,
Copyright(C) 2004 Mentor Graphics Corp

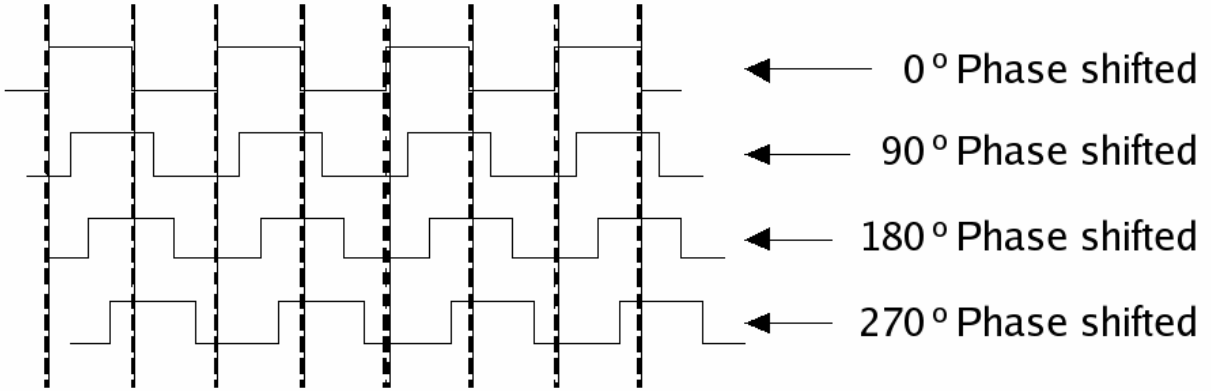
- *Frequency synthesis*: Allows the external clock frequency to divided or multiplied.



The Design Warrior's Guide to FPGAs,
ISBN 0750676043,
Copyright(C) 2004 Mentor Graphics Corp

Clock Trees and Clock Managers

- *Phase shifting*: Phase shifting a clock with respect to another adds delay to it.



The Design Warrior's Guide to FPGAs,
ISBN 0750676043,
Copyright(C) 2004 Mentor Graphics Corp

Common phase shifts are 120 and 240 degrees (for 3-phase clk schemes) and 90, 180 and 270 degrees for 4-phase schemes.

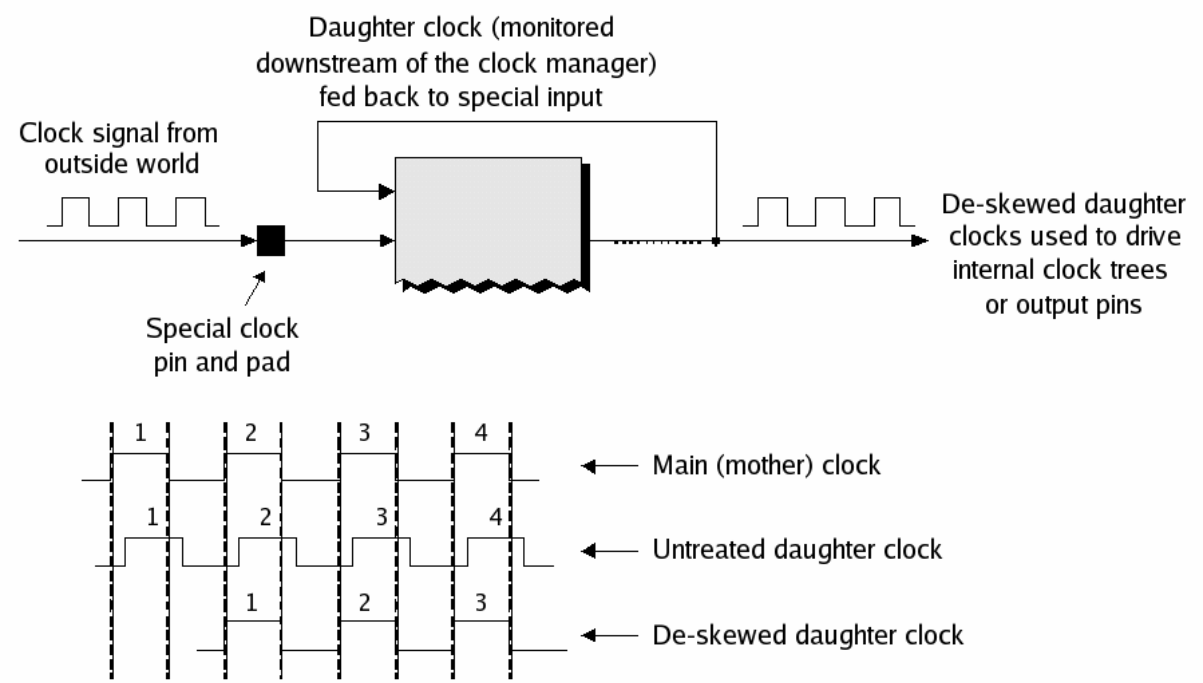
Some clock managers allow *any* value to be set for each daughter clk.

- *Auto-skew correction*: Allows for automatic correction of delays introduced by the clock manager and interconnect.

  This is accomplished by "feeding back" the daughter clock to be corrected.

Clock Trees and Clock Managers

- *Auto-skew correction* (cont.)



Skew in the daughter is removed by comparing it with the external clk and delaying it until the edges re-align.

Some clock managers are based on *phase-locked loops* (PLLs) and others on *digital delay-locked loops* (DLLs).

Trade-offs include precision, stability and noise insensitivity.

General-Purpose I/O

With 1000 or more pins on today's FPGAs, special packages that allow for *area arrays* are used (2-dimensional distribution of pads across the chip).

For simplicity, let's assume the older style of packaging with peripheral I/Os.

Configurable I/O standards:

Given the wide variety of standards that exist for representing logic 0 and logic 1 at the board level, FPGA have *general purpose I/Os*.

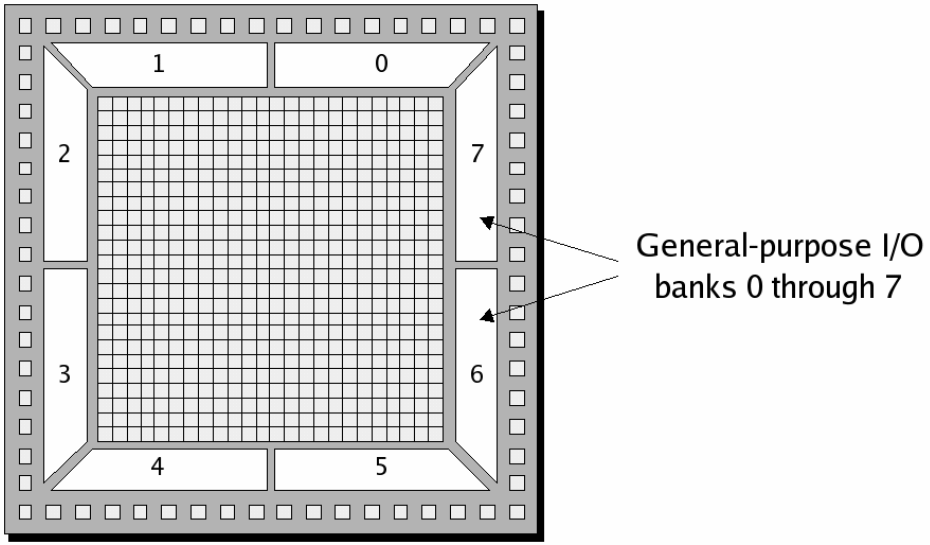
They can be configured to accept and generate signals conforming to any one of these standards.

The I/Os are organized into a set of *banks*, each of which can be configured individually to support a particular I/O standard.

This increases the versatility of the FPGA and allows it to act as an interface between different I/O standards.

General-Purpose I/O

Configurable I/O standards:



Configurable I/O impedances:

FPGAs I/O pins can be configured with specific impedances (terminating resistances) to cancel signal reflections.

Signal reflections result from very fast edge rates, that cause signals to bounce back and forth across the wires connecting chips.

Core vs. I/O Supply Voltages

As feature size reduces in new technology generations, so does the *core* supply voltage.

Year	Supply (V)	Tech node
1998	3.3	350 nm
1999	2.5	250 nm
2000	1.8	180 nm
2001	1.5	150 nm
2003	1.2	130 nm

The days when every chip used a 5 V core supply voltage are over.

In order to accommodate the variety of voltages, each I/O bank has its own supply pins, that are separate from the *core* supply pins.

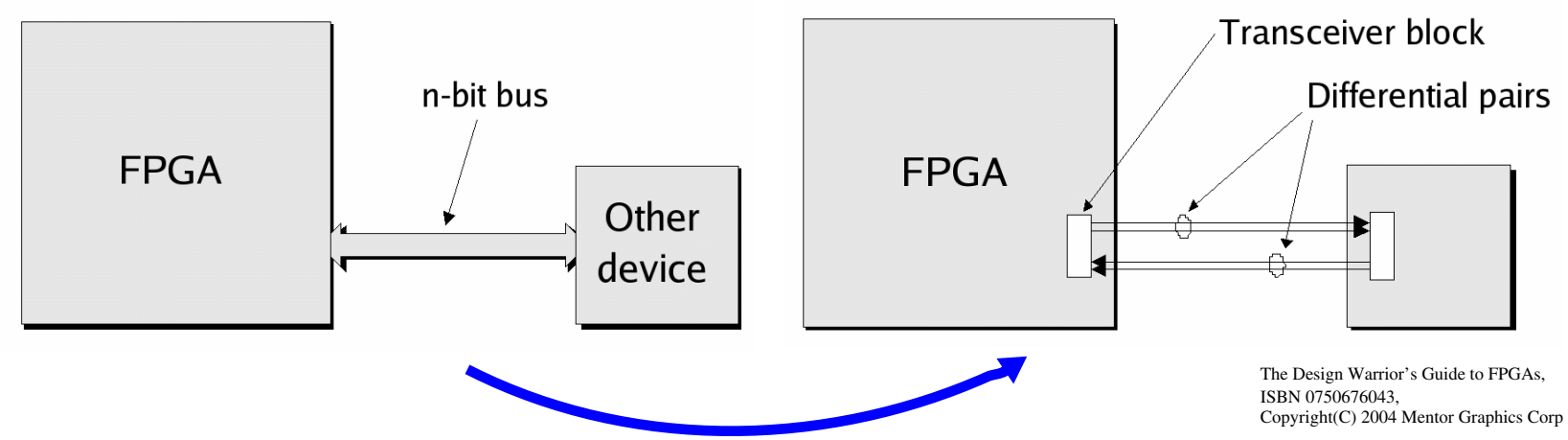
Gigabit Transceivers

The traditional way of moving data around is to use a *bus*.

As bus widths increased from 8 to 64 bits, routing buses became a significant challenge (matched impedances and signal integrity issues).

Gigabit Transceivers

In order to overcome this problem, FPGAs started including *hard-wired gigabit transcievers*.



The Design Warrior's Guide to FPGAs,
ISBN 0750676043,
Copyright(C) 2004 Mentor Graphics Corp

These blocks use one pair of **differential signals** (signals with opposite polarities) for *transmit* (TX) and one pair for *receive* (RX).

These serial communication channels accomplish what a bus accomplished by operating at very high frequencies (gigabits/sec).

It is common to bundle 4 such *transceivers* in each block.

Hard IP, Soft IP and Firm IP

Hard IP refers to pre-implemented blocks, such as microprocessor cores, gigabit interfaces, multipliers, adders, MAC functions, etc.

They are optimized for speed, power and area.

Each FPGA vendor offers its own combinations of such blocks.

Soft IP refers to a *source-level library* of high-level functions that can be included in the user's designs.

These functions are usually represented in an HDL at the *register transfer level* (RTL), and are realized in the PLBs during synthesis.

Firm IP refers to functions that have already been optimally mapped, placed and routed into groups of PLBs (possible combined with some hard IP).

It is often difficult to decide whether a block should be implemented as *hard IP* or as *soft* or *fi rm IP*

Hard IP, Soft IP and Firm IP

Multipliers, adders and MACs are common enough to avoid debate.

However, blocks such as a *PCI interface* are more difficult to decide on.

If you use it, great. If not, it only occupies space and consumes power.

Therefore, adding something too specialized funnels the product into a niche market.

System Gates vs. Real Gates

Different vendors have different implementations of functions, which have varying numbers of transistors.

In order to make it easier to compare relative capacity and complexity of 2 chips, ASIC vendors measure size in terms of *equivalent gates*.

Here, each function is assigned an equivalent gate value, e.g., "function A equates to 5 equivalent gates."

System Gates vs. Real Gates

Complexity is then measured by summing up all the *equivalent gate* values.

Unfortunately, what constitutes the definition of an *equivalent gate* varies between vendors.

One common convention is for a 2-input NAND function to represent **one equivalent gate**.

Another convention is for a vendor to define an *arbitrary number* of transistors as an equivalent gate.

W.r.t. FPGAs, vendors need a way of telling a customer with an 500,000 equivalent gate ASIC that his design will or will not fit onto an FPGA.

The problem here is that a 4-input LUT can represent between 1 and >20 2-input primitive logic gates.

FPGA vendors (in the early '90s) starting talking **system gates**.

System Gates vs. Real Gates

Unfortunately, there is no clear definition of a *system gate*.

Some say divide the number of system gates by 3 to get equivalent gates, while others say divide by 5.

The task of equating *equivalent gates* to *system gates* remains difficult at best.

Add to the mix *embedded RAM*, and hard-cores such as multipliers and adders, and the task becomes nearly impossible.