UMBC

AN HONORS UNIVERSITY IN MARYLAND

Department of Computer Science and Electrical Engineering

CMPE 415 Programmable Logic Devices

FPGA Technology II

Prof. Ryan Robucci

Some slides (blue) developed by Jim Plusquellic
Some images credited to book: Maxfield, Clive. The design warrior's guide to FPGAs: devices, tools and flows. Elsevier, 2004.

ASICs

ASIC (gate array, etc.)

Four main classes exist today, in order of increasing complexity:

- Gate arrays
- Structured ASICs
- Standard cell devices
- Full-custom chips

Full-custom

In the early days, only two classes of chips existed.

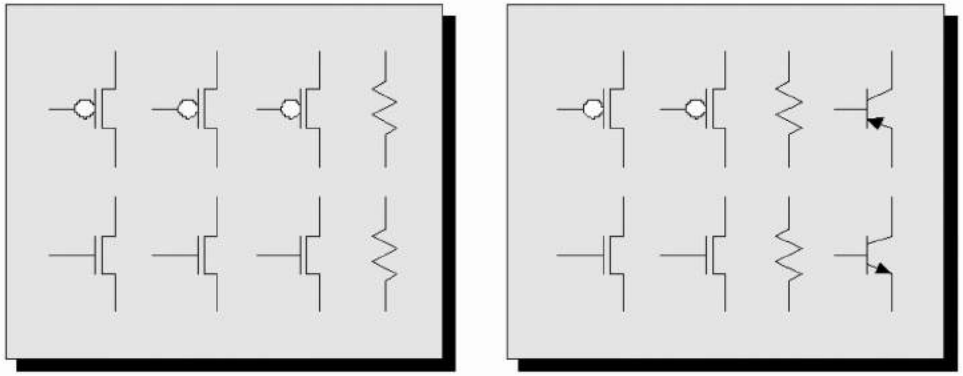
- Standard off-the-shelf components
- Full-custom ASICs (such as microprocessors)

For the full-custom class, nothing is predefined, not even standard logic gates.

All wires and gates are hand-crafted individually, and optimized for speed, area and power.

ASIC (gate array, etc.)

Gate arrays are based on the idea of a *basic cell* consisting of a collection of unconnected transistors and resistors.



(a) Pure CMOS basic cell (b) BiCMOS basic cell

The ASIC vendor *prefabricates* silicon chips containing arrays of these **basic cells**.

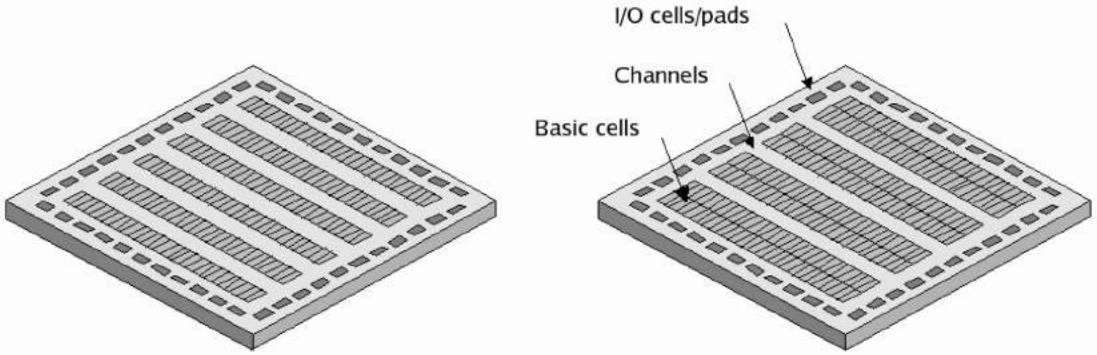
Channels are typically provided between rows or columns of these arrays for routing.

Alternative, *sea-of-gates* arrays do not have routing channels.

The vendor also provides a *cell library* (a set of basic logic gates, MUXs, etc).

Gate Arrays: Cheaply build arrays of FETs but don't connect them until later.

ASIC (gate array, etc.)



(a) Single-column arrays (b) Dual-column arrays

The Design Warrior's Guide to FPGAs, ISBN 0750676043,

Designers use the latter and generate a *netlist*.

Special mapping, placement and routing CAD tools are used to assign the logic gates to *basic cells* and to design the interconnect.

The output of this process are *photo-masks* that define the metalization layers.

Since the transistors and other components are **prefabricated**, there is a considerable cost savings.

The main drawback is *resource under-utilization* and *less-than-optimal routing* (because of routing constraints).

Vendors provided well-characterized blocks and supported tool flows or rapidly "designed" IC in-house

Standard Cell Devices

Became available in the early '80s to address the problems with gate arrays.

Similar to gate arrays, the ASIC vendor defines the *cell library.*

Designers generate a *gate-level netlist* using CAD tools (similar to gate arrays) but none of the components are prefabricated.

P&R tools *place* and *route* each of the gates and optimize for area and delay.

The standard cells themselves are designed as *constant height cells*, which simplifies their placement (PWR and GND are connected via abutment).

The output of the process is a *complete* set of photomasks.

Pre-designed Blocks:

The vendor supplies *hard-macro* and *soft-macro* libraries, which include elements such as processors, comm. functions, RAM, ROM, etc.

Designers also have the option of purchasing blocks of **intellectual property** (IP).

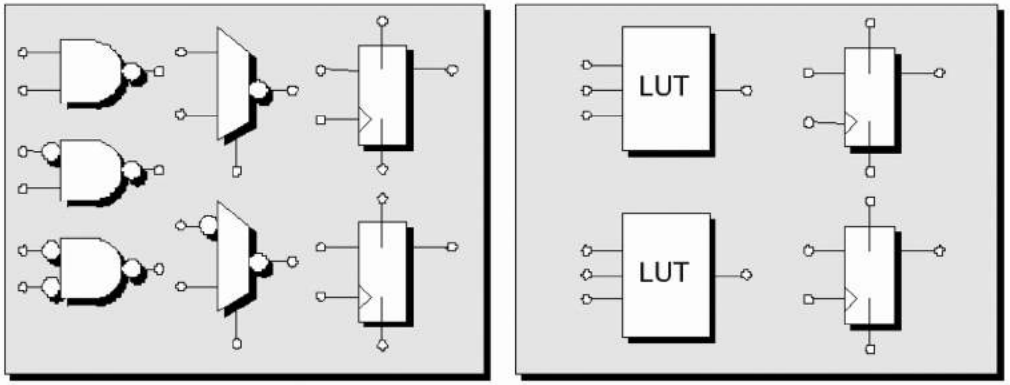
Structured ASICs

Entered the scene in the early '90s but were not accepted until '03.

ASIC manufacturers were looking for ways to reduce ASIC *design costs* and *development times*, without reverting to gate arrays.

The basic element is called a *module* or *tile*.

It contains a mixture of prefabricated generic logic (implemented as gates, MUXs or lookup tables), registers and some local RAM.



(a) Gate, mux, and flop-based (b) LUT and flop-based

An array (sea) of these elements are prefabricated.

Structured ASICs

Peripheral elements are also prefabricated, and include RAM blocks, clock generators, boundary scan logic, etc.

Similar to gate arrays, the chip can be customized using only metalization layers.

Since structured ASIC tiles are more complex than gate arrays, *most* of the metalization layers are predefined.

Most require only 2 or 3 layers to be customized, in the extreme, one requires the definition of only a single *via* layer.

This saves considerable time and cost since only a couple photo-masks need to be designed.

The disadvantages include about **3X** more area and **2-3X** more power, when compared with a standard cell chip.

Among all the ASIC options: Tradeoff design time and cost versus level of customization and performance.

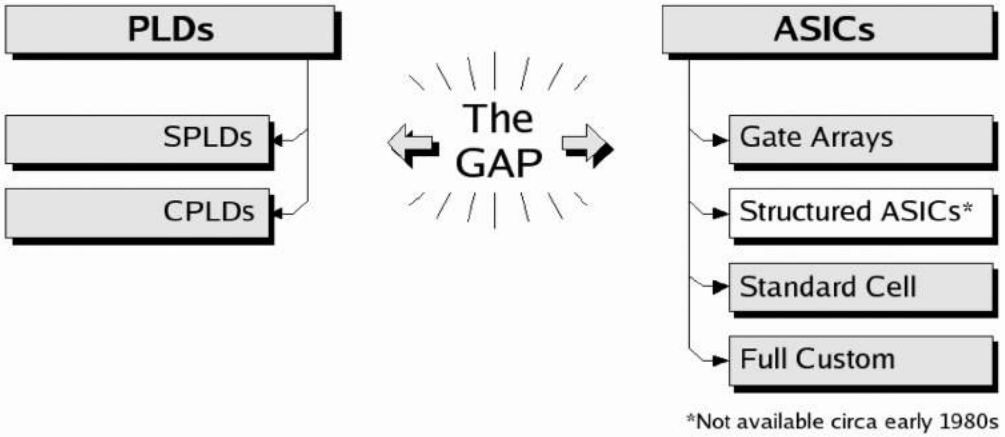
FPGAs

FPGAs

In early '80s, a gap emerged in the digital IC continuum.

At one end, SPLDs and CPLDs provided high configurability, fast design and modification times, but supported only small to moderate functions.

At the other end, ASICs supported large complex designs but were immutable once fabricated, expensive and time-consuming to design.

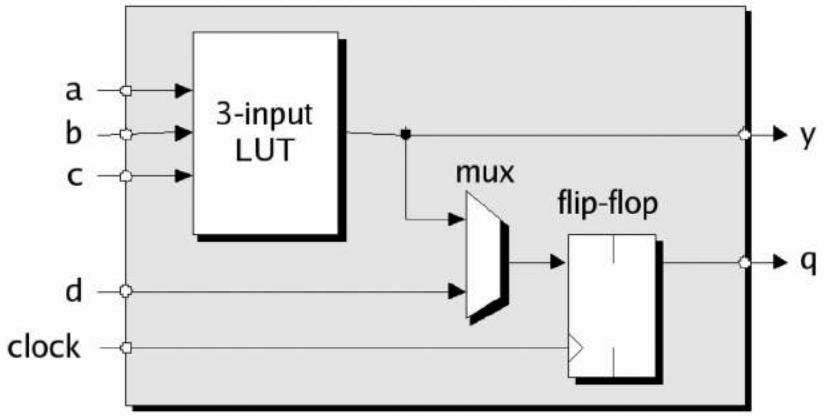


Xilinx developed and made available in '84 a new class of IC called the **FPGA** to fill the gap.

FPGAs

The first FPGAs were based on CMOS and used SRAM cells for configuration.

The early chips used an array of **programmable logic blocks** (PLBs), which comprised a 3-input *lookup table* (LUT), a register and a MUX.



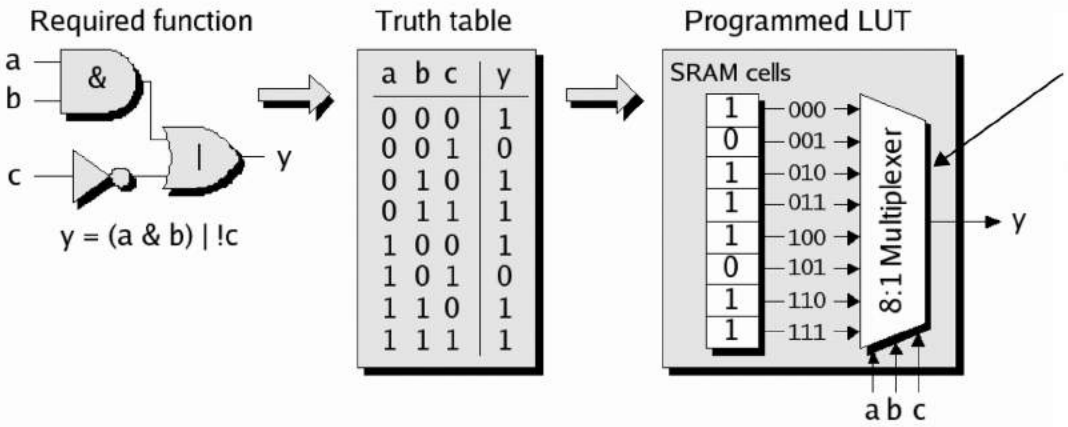
Each PLB can be programmed individually to perform a unique function.

The FF can be triggered by a positive or negative-going clk.

The MUX allows selection of the LUT output or an external input.

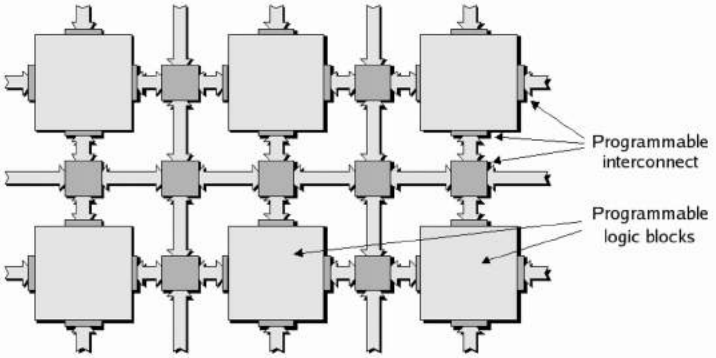
FPGAs

The LUT can implement any 3-input logic function.



(Actual implementation does not use a MUX here -- more on this later).

The FPGA architecture consisted of a 2-dimensional array of PLBs separated by a *sea* of programmable interconect.



FPGAs

Todays FPGAs are much more complex (to be discussed).

For example, in addition to the local interconnect, an FPGA typically has a **global** (high-speed) **interconnection** network.

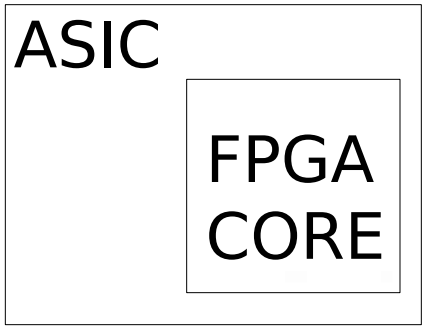
This allows signals to cross the chip without having to pass through local switching elements.

FPGA-ASIC hybrids

Although it doesn't make sense to embed an ASIC inside an FPGA, it is meaningful in the other direction, i.e., embedded FPGA cores.

This is useful for *platform design*, a term used at the board level to refer to a design from which **multiple products** can be derived.

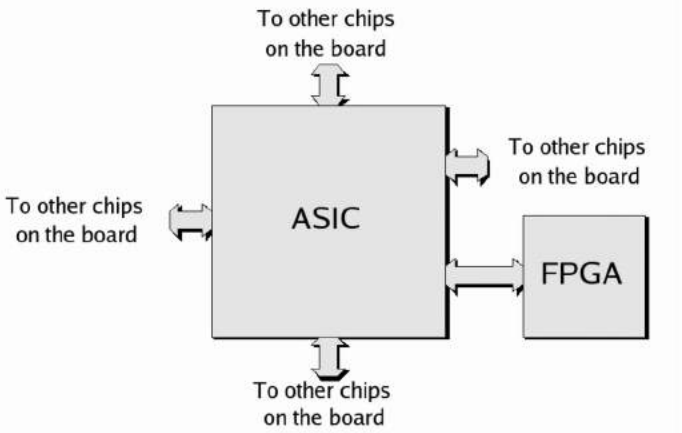
Here, the *platform* is the ASIC and the embedded FPGA allows it to be customized for a specific application.



FPGA-ASIC hybrids

Another driver is provided by the increasing number of incidences of FPGAs being used to augment ASIC designs.

This has traditionally been accommodated at the board level.



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

Here, the designers of the ASIC *off-loads* to the FPGA any part of the ASIC design that is subject to modifications or enhancements.

Board level realization incurs a performance penalty because signals must travel off-chip.

Embedding the FPGA solves this problem.

FPGA-ASIC hybrids

Designing these hybrids however is challenging because ASIC and FPGA design tools/flows are significantly different underneath.

ASIC are **fine-grained** because they are implemented at the *primitive logic gate* level.

FPGAs are **medium-grained** (or coarse-grained according to others) because they are realized using higher-level blocks (PLBs).

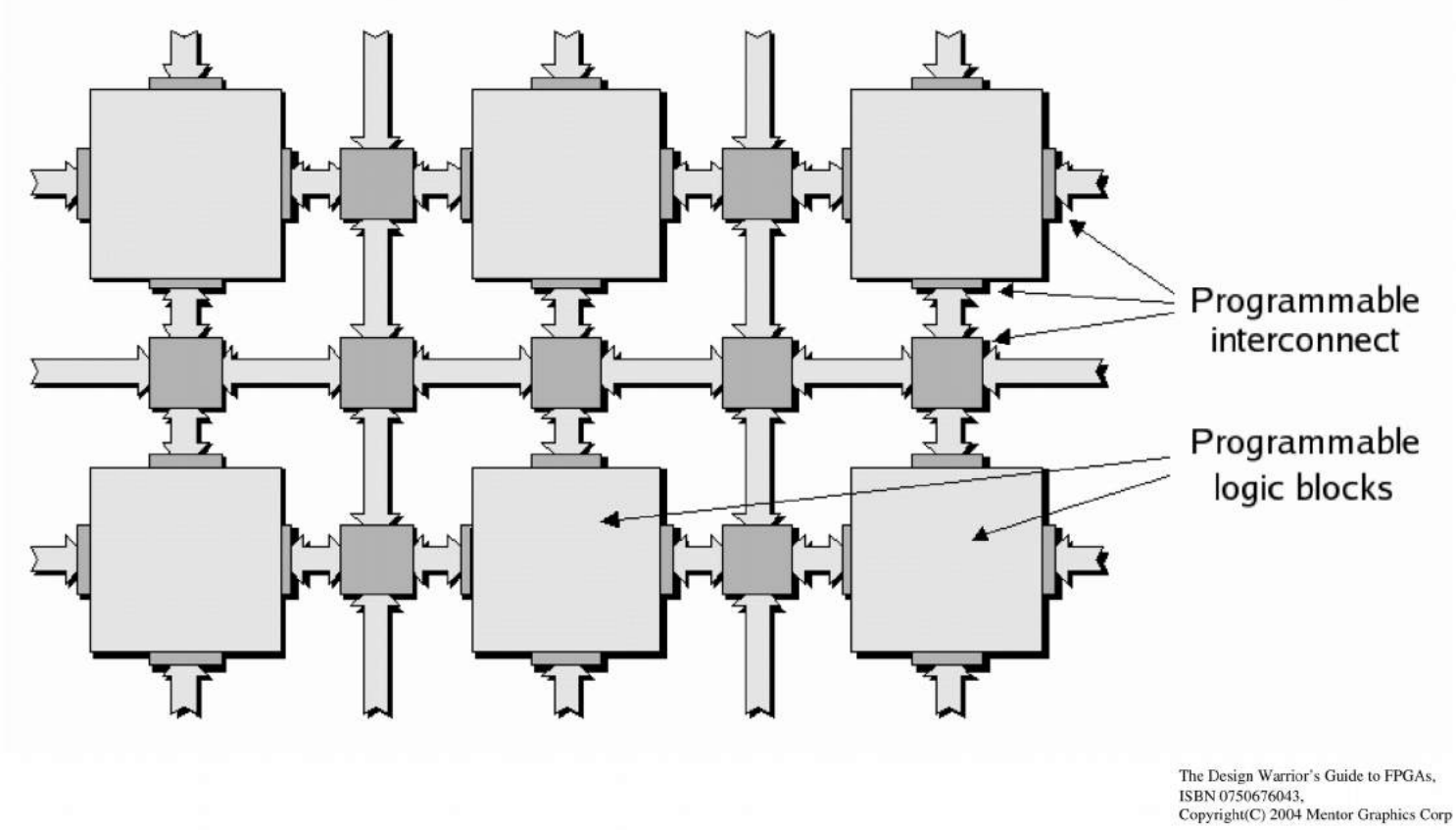
The CAD tools that do the synthesis and P&R need a consistent view (fine-grained or medium grained) of the world in order to do a good job.

Structured ASIC and FPGAs however are both medium-grained and therefore can enjoy a unified tool and design flow.

That is, the same block-based synthesis and P&R engines can be used for both the ASIC and FPGA portions.

Fine-, Medium-, and Course-Grained Architectures

Basic architecture consists of a large number of PLB islands embedded in a sea of programmable interconnect.



Fine-, Medium-, and Course-Grained Architectures

"Level of granularity", when used in the context of an FPGA refers to the complexity of the PLB.

The PLBs of **fine-grained** architectures can only implement simple functions, e.g., 3-input logic gate or storage element.

Good for glue logic, state machines, systolic algorithms (massively parallel), and traditional logic synthesis.

The PLBs of **medium-grained** architectures include more logic and more functionality, i.e., a 4-input LUTs, 4 MUXs, 4 D-FFs + fast carry logic.

This helps with the interconnect problem, e.g., more "compute power" per wire dedicated for interconnections.

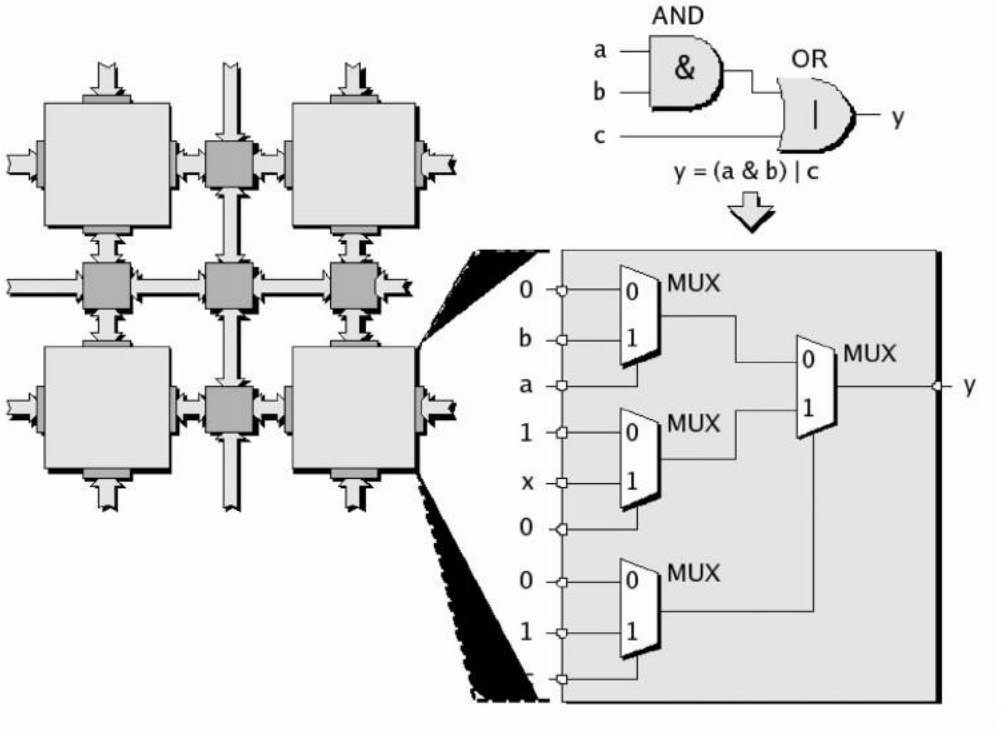
Large-grained architectures incorporate FFT engines and microprocessor cores.

Fine-grained architectures of the mid-90's gave way to the *medium grained* architectures.

MUX- vs. LUT-based Logic Blocks

There are 2 basic flavors of PLBs for *medium-grained* architectures, *multiplexer* (MUX) and *lookup table* (LUT).

In the MUX-based version, each input can be programmed with a logic 0, 1, or the true or inverted version of a variable.

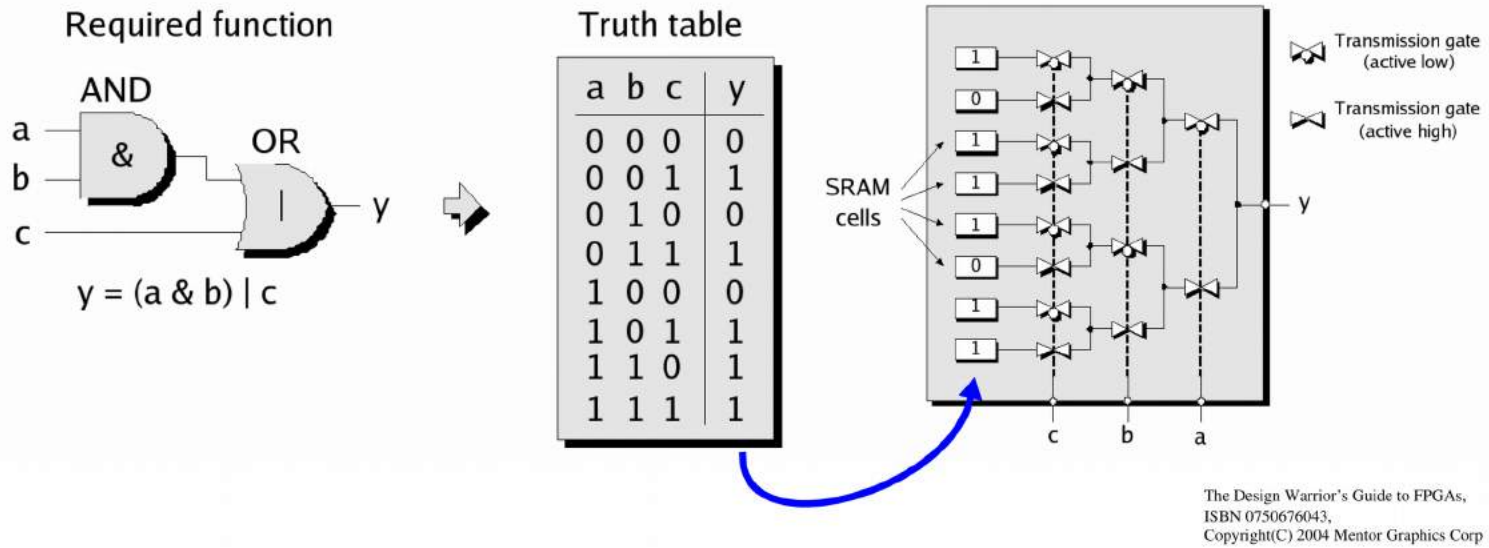


Any logic can be built from only muxes when "1" and "0" inputs are also available.

For review: implement (a ^ (~b)) using a 2-input mux

MUX- vs. LUT-based Logic Blocks

Most FPGAs today are LUT-based -- here, the input signals are used as a pointer into a lookup table.



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

Input signals can be decoded using a hierarchy of *transmission-gate* MUXs.

Transmission gates *pass* the value on their inputs or are **high-impedance**.

Note that the diagram does not show the serial connection of the cells (scan chain) for simplicity.

Transmission gates are basically switches implemented with FETs.

MUX- vs. LUT-based Logic Blocks

Larger LUTs are possible, e.g., 3-, 4-, 5- and 6-input versions.

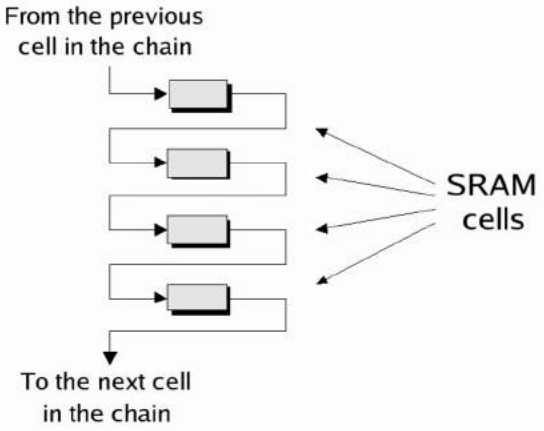
Every time an input is added, the size of the table doubles.

4-input versions are believed to provide the optimal balance today.

Basically, You can use memory functionally as memory instead of as logic

Some vendors allow the 16 cells in the LUT to play the role as a *16x1 RAM*, and sets to be strewn together to form larger RAMs.

Some vendors allow the 16 cells of the LUT to be decoupled from the larger chain and used as a shift register.

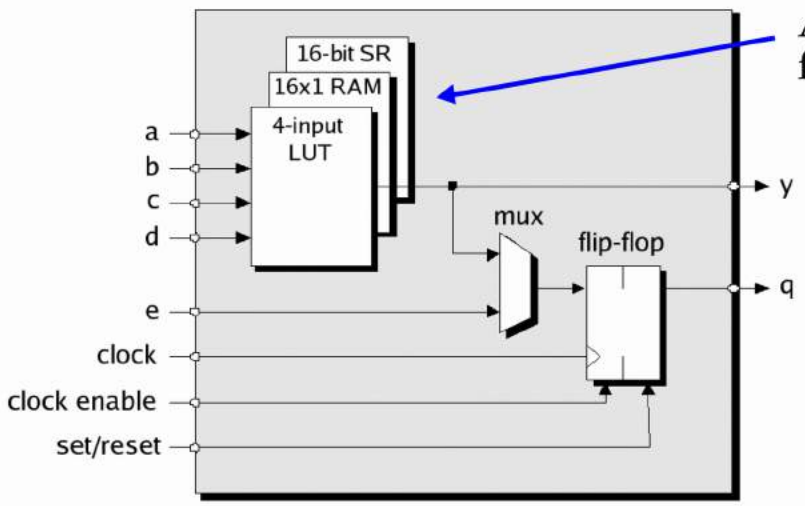


The Design Warrior's Guide to FPGAs,
ISBN 0750676043,
Copyright(C) 2004 Mentor Graphics Corp

Moving up the internal hierarchy

Terminology

Xilinx calls them ***logic cells*** **(LC)** while Altera calls them ***logic elements*** **(LE)**.



As described, LUT is more versatile for feature expansion than MUX

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

A *slice* is defined as 2 LCs.

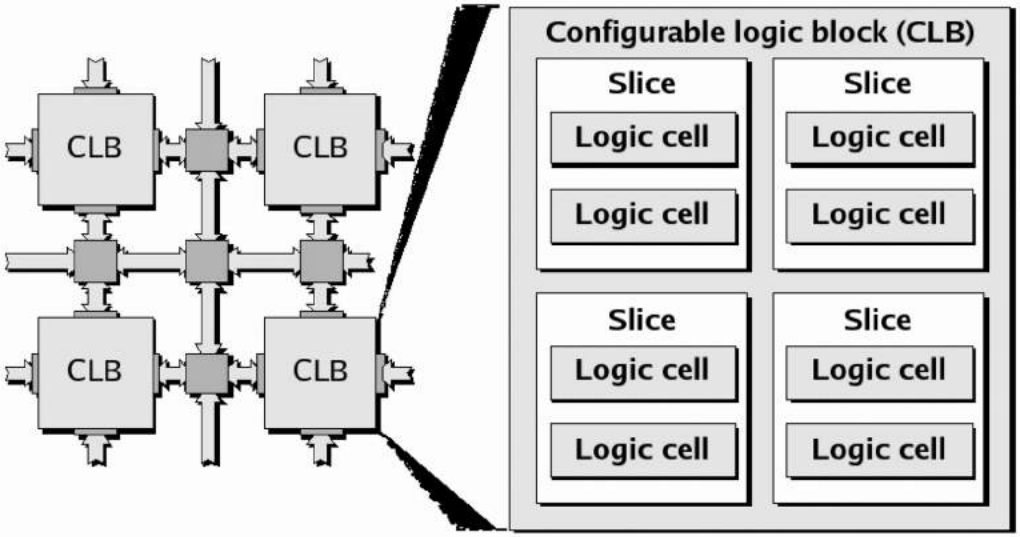
Each instance has its own inputs and outputs but the *clock*, *clock enable*, and *set/reset* signals are common.

A *configurable logic block* (CLB, Xilinx) or *logic array block* (LAB, Altera).

CLBs consist of 2 or 4 *slices*, and conform to the islands shown earlier.

Terminology and Hierarchy

The CLB also has some fast interconnect (not shown), that is used to connect neighboring slices.



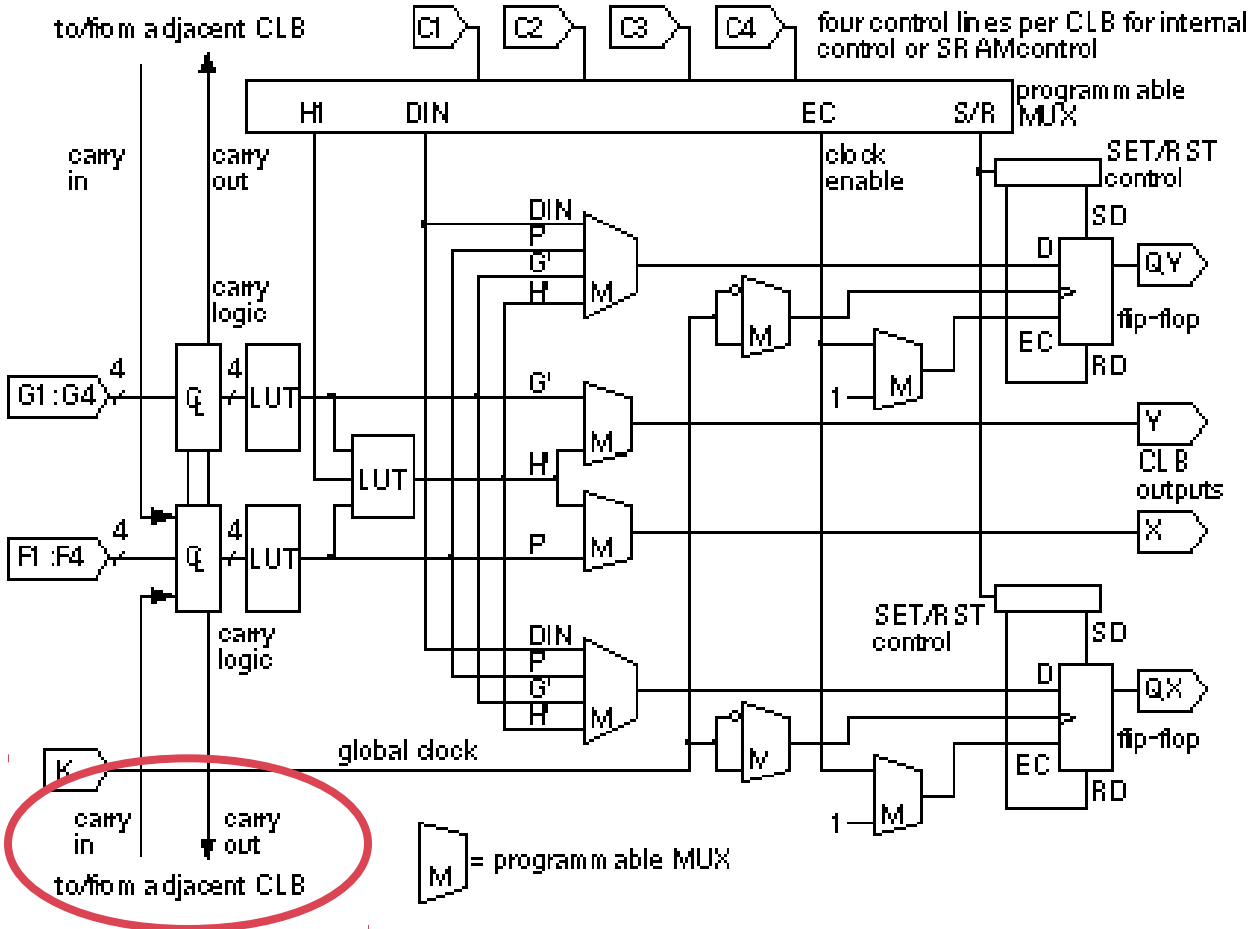
The organization of *LC* -> *Slice* -> *CLB* is complemented by an equivalent hierarchy in the interconnect.

That is, fast interconnect between LCs in a slice, slightly slower between slices in a CLB, followed by the interconnect between CLBs.

FPGA IC designers have to design local and global routing (and switches) and decide how much space to allocated to each

Complex Logic Block Cells

- the CLB used in the XC4000 series of Xilinx FPGAs. This is a fairly complicated basic logic cell containing 2 four-input LUTs that feed a three-input LUT. The XC4000 CLB also has special fast carry logic hard-wired between CLBs. MUX control logic maps four control inputs (C1–C4) into the four inputs: LUT input H1, direct in (DIN), enable clock (EC), and a set / reset control (S/R) for the flip-flops. The control inputs (C1–C4) can also be used to control the use of the F' and G' LUTs as 32 bits of SRAM.



“Hard” Hardware vs “Soft” Hardware

- Soft IP or a soft hardware core is “compiled hardware” programmed into the FGPA
- Sometimes it is not possible or best to implement required hardware as a “soft” hardware core
- Factors for inclusion of hard IP core as a market advantage:
  - Does a Hare IP represent significant performance advantage over soft IP? Sometimes implementation in the FPGA fabric is too big, slow, etc…
  - Is a soft implementation not possible and is outside the bound of performance programmable fabric can achieve: e.g. high-speed IO
  - Would multiple customers would utilize it? meaning sufficient market demand
  - Cost – what area does is consume and how much does it negatively impact the overall FPGA performance?
- Sometimes, we will characterize specialized hardware as a feature added to the programmable fabric (e.g. carry-chain adder support) or a separated hardware like high-speed IO
- A common characteristic of any specialized hardware on FPGAs is some level of configuration. e.g. high-speed IO hardware should at least support different rates, voltage levels, etc….

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 *firm 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.

FPGA Additional Features and Characteristics

Assuming 4 slices per CLB, several RAM configurations are possible.

- **Single-port 16x8, 32x4 bit, 64x2 or 128x1 bit RAM**
- **Dual-port 16x4, 32x2, 64x1 bit RAM (reads and writes occur through individual ports)**

The LCs of the slices and slices of the CLB contain special interconnect to allow the *16 bit shift registers* of each LC to be combined into a *128 bit chain*.

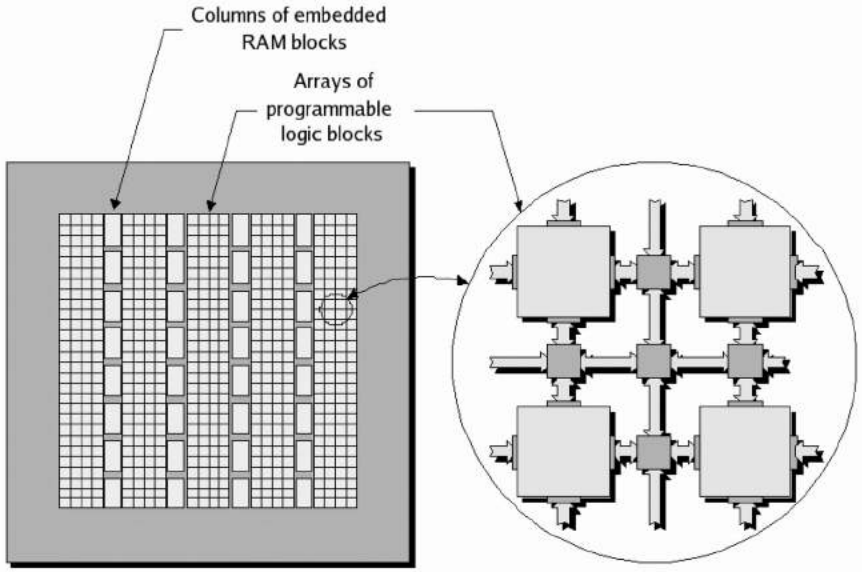
The LCs also include *special carry logic* (for fast carry chains), and dedicated interconnect between LCs in a slice, between slices and between CLBs.

These features boost the performance of logical functions, e.g., counters and adders.

Many applications require memory, so FPGAs now include *embedded RAM* called **e-RAM** or **block RAM**.

FPGA Additional Features and Characteristics

Some FPGAs include these blocks around the periphery of the chip, others distribute the blocks, while others organized it in columns.



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

Each block can hold up to 10K bits, and each chip may contain hundreds of these *eRAM* blocks, for storage capacity up to several million bits.

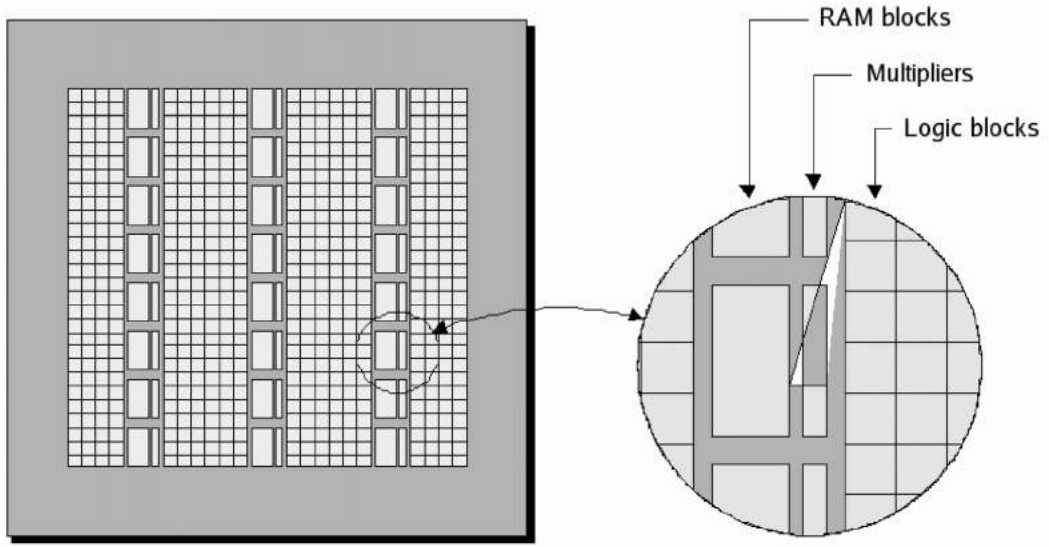
The *eRAM* blocks can be used separately or combined, and are useful for implementing single/dual-port RAMs, FIFO functions, state machines, etc.

FPGA Additional Features and Characteristics

Some functions, like multipliers, are slow if implemented by connecting LCBs together.

Since these functions are common, many include special hardwired multiplier blocks.

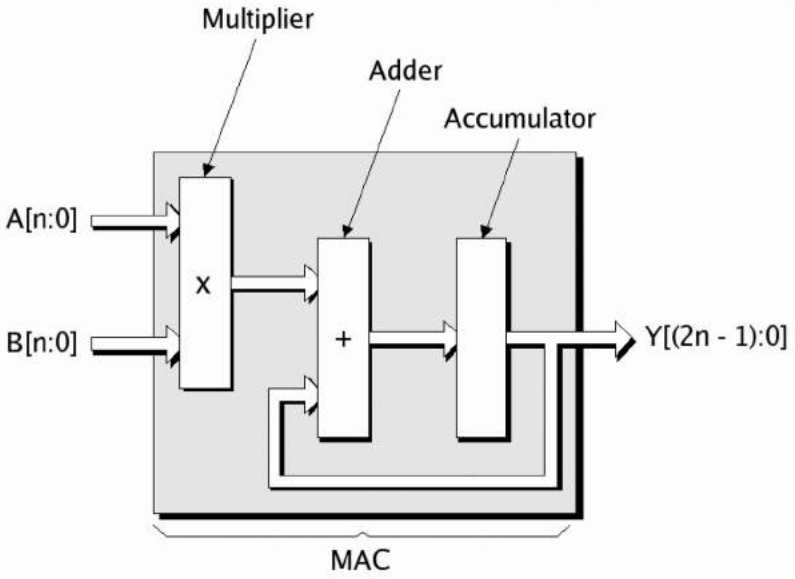
They are often located near the *eRAM* blocks.



FPGA Additional Features and Characteristics

Some FPGAs offer dedicated adder blocks.

These are useful in DSP operations called *multiply-and-accumulate* (MAC).



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

This operation consists of multiplying two numbers and storing the result in as a running total (in the *accumulator*).

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.

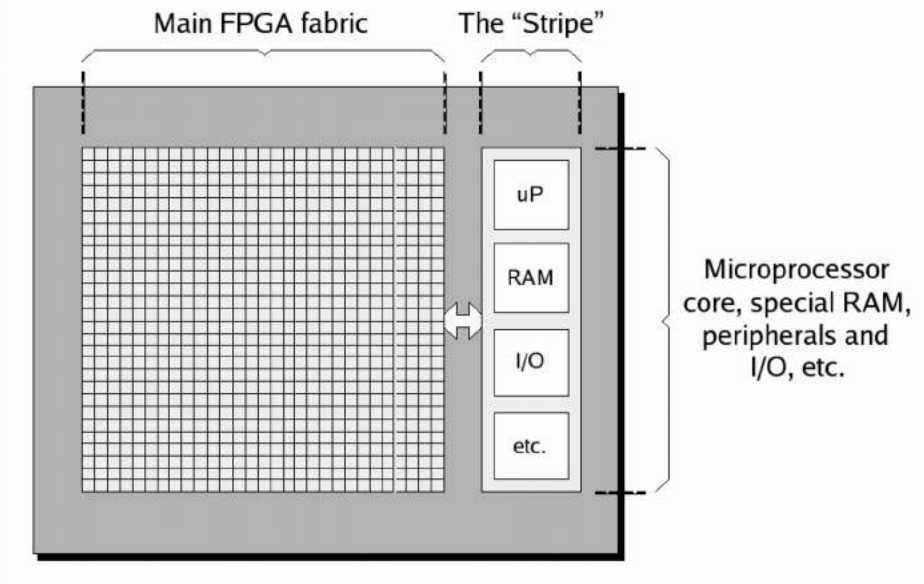
Until recently, these appeared as discrete chips on the PCB -- now they appear as embedded **microprocessor cores** within the FPGA.

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)

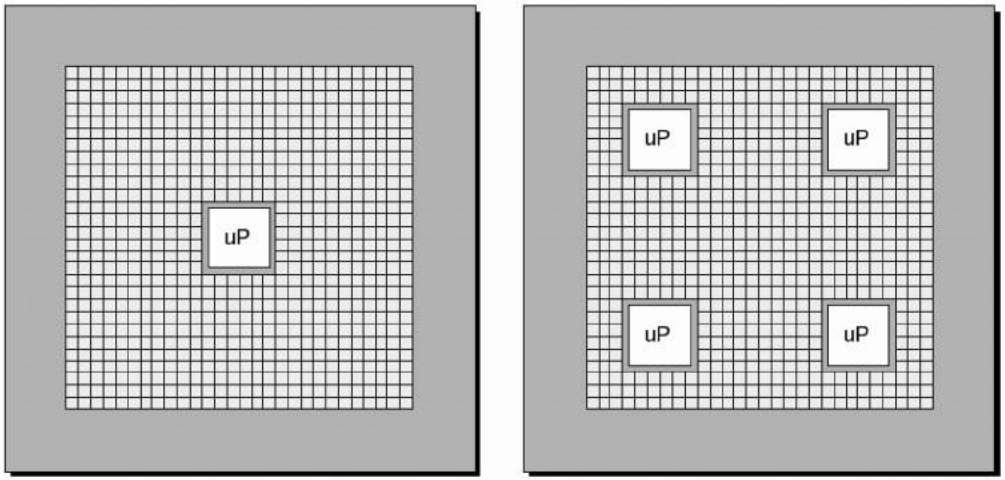
- In a strip (called the Stripe) to the side of the FPGA fabric (cont).

  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)

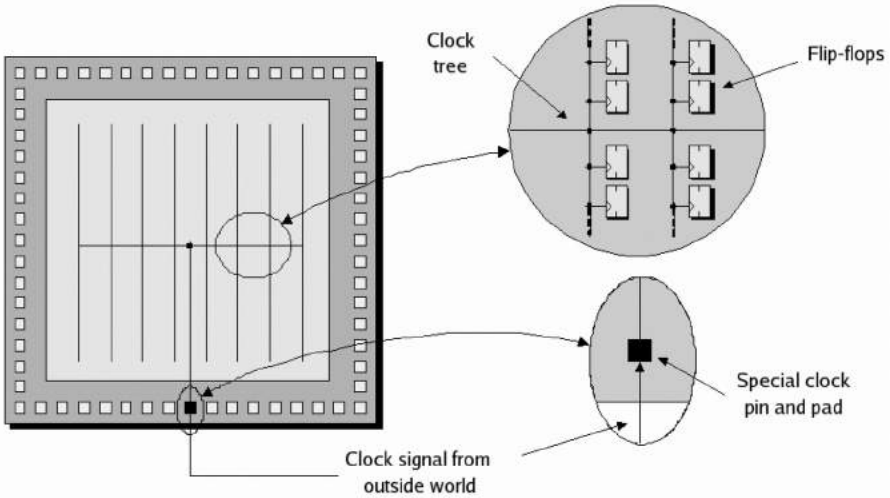
For *soft microprocessor cores*, a group of PLBs are configured as the micro.

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

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*.

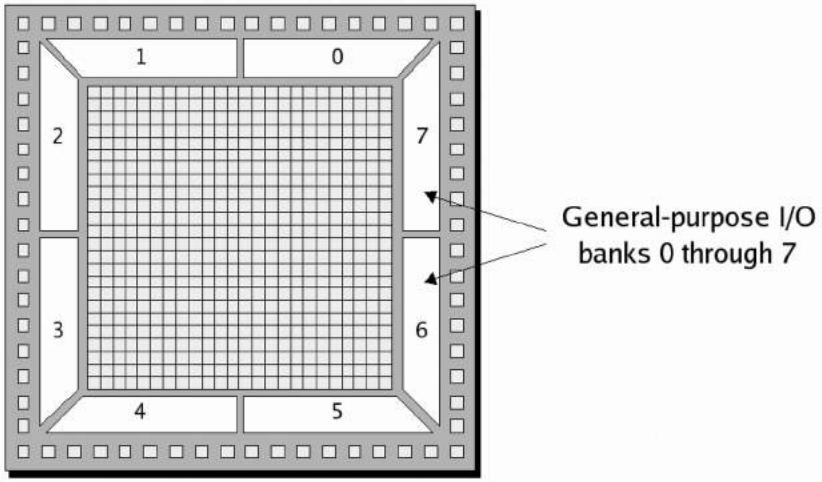
They can be configured to accept and generate signals conforming to any one of these standards.

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*.

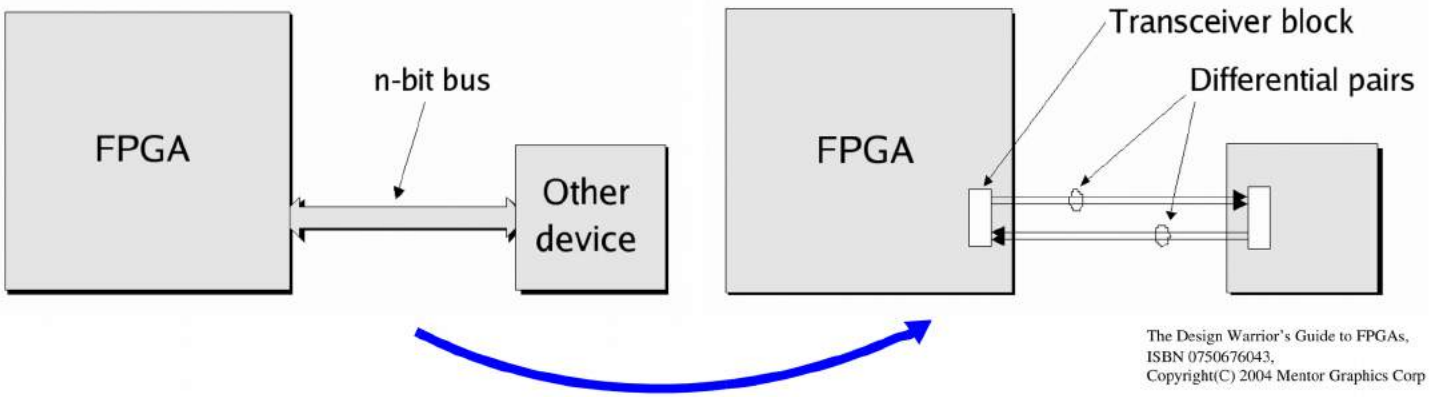
As bus widths increased from 8 to 64 bits, routing buses became a significant challenge (matched impedances and signal integrity issues).

High-Speed Serial vs. Parallel

- High speed communication requires
  - fast signal transitions which create reflections in the same line and interference with other signals
    - Mitigation requires controlled impedance which means the lines must have very consistence physical dimensions and precise impedance at one or both ends
  - Synchronization between signals
    - all lines must have matched electrical delays
- So then, why not use parallel and run things slower so that you don't have to worry about these things?
  - At some point it becomes necessary to design for high speed signals. Past that point, it can be easier to work with fewer signals.
  - Cabling becomes epically expensive for high-speed parallel

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.

Gate Count / “Design Size“

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.

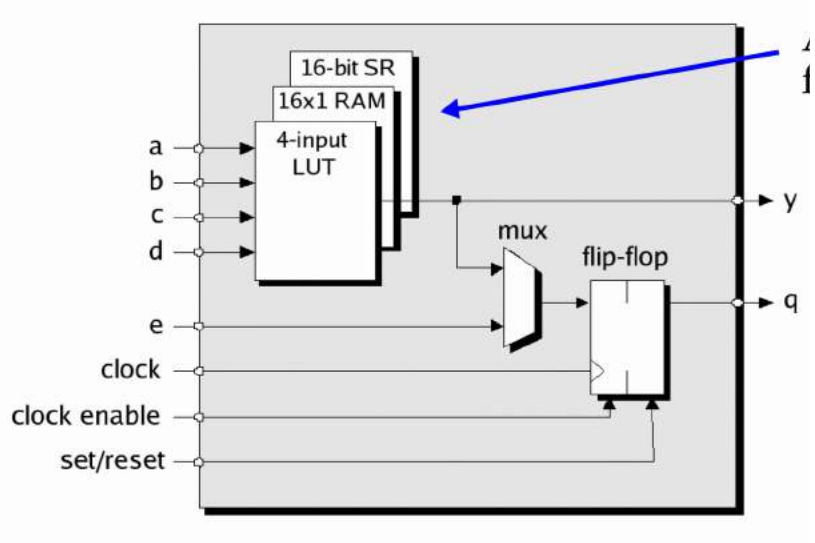
Points to consider when estimating design "size":

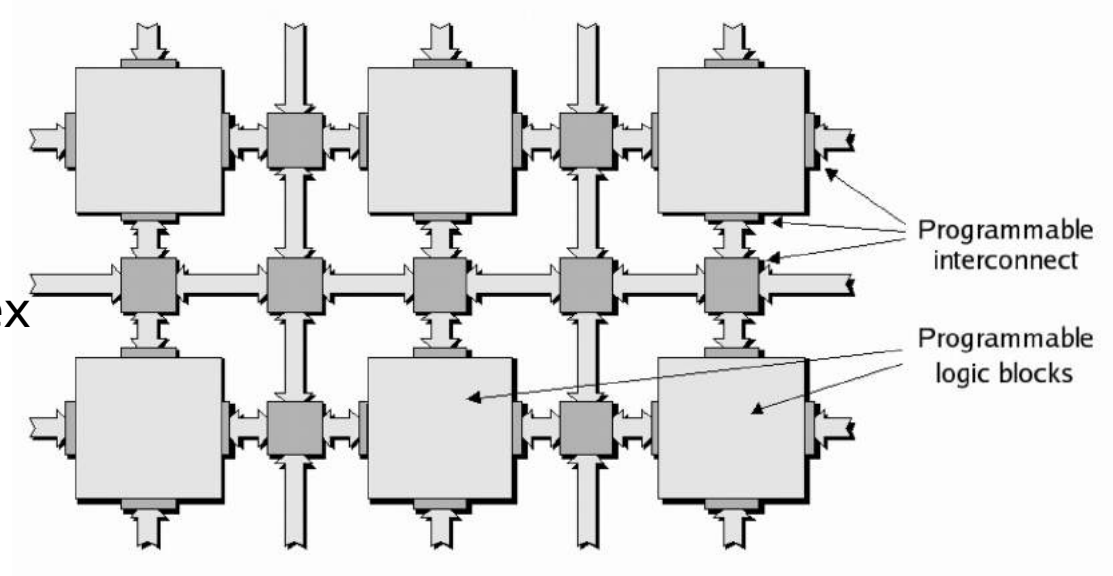
- Number of logic cells/logic elements/things equiv. to 4-input LUTs+LATCHES+ETC
- Number and size of embedded RAM
- Number and size of embedded multipliers
- Number and size of embedded adders (accumulators)
- Number and size of embedded MAC
- Number of any other special hardware

What needs to be configured?

- IO pin configuration
- Routing
- Logic Cells
  - LUT contents, configuration
  - mux controls and io connections
  - FF/latch sensitivity (rising or falling edge or high or low level
  - reset values of FF
- Any configurations bits for Hard IP
  - RAM – init to 1s or 0s, typically don't set complex initial contents through configuration stream,

    maybe later through JTAG





Configuration Files and Cells

The end result (as you know) of the design flow is a *configuration file* or *bit file* that is used to program the FPGA.

For SRAM-based, EEPROM and FLASH FPGAs, the *bit file* contains both *configuration data* and *configuration commands*.

The commands tell the FPGA what to do with the data.

For anitfuse, the file contains only configuration data.

Configuration cells are present in the device to allow the interconnect, I/O and other features of the FPGA to be programmed.

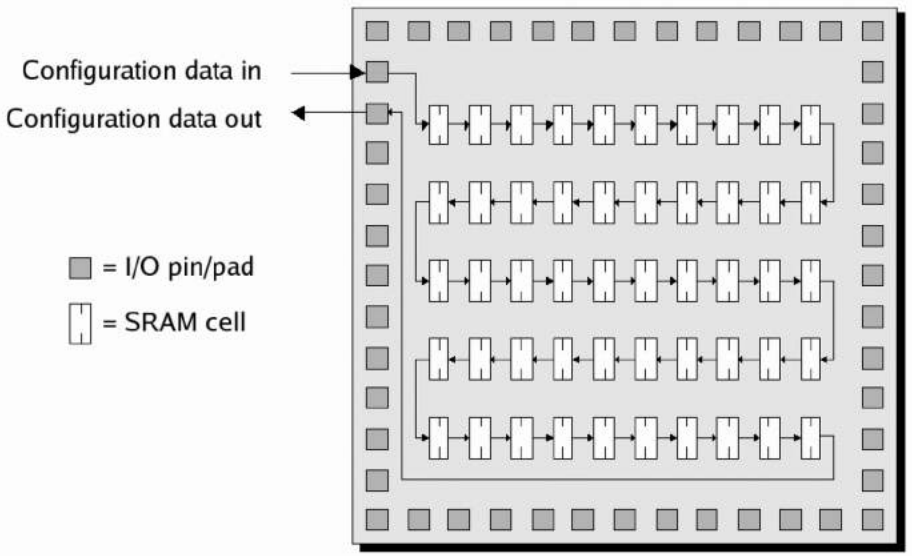
For example, within the PLB, the MUX's *select input* needs a configuration cell.

The register needs configuration cells to specify whether it is to act as a FF or a latch, whether it latches using a positive- or negative-going clk, and whether it is initialized to a 0 or a 1.

Configuring the FPGA

A 4-input LUT contains 16 configuration cells.

The configuration cells are typically connected in a long *scan chain*.



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

The *scan chain* (when programmed in this mode) has an input and an output.
The output is used if multiple FPGAs are *daisy-chained*.

Embedded RAMs are implemented as latches and are part of the scan chain.

Configuring the FPGA

Note that this is a simplistic view of the FPGA's internal organization.

In reality, the scan chain is made from latches, not FFs.

Latches are **half** the size - saves a lot of real estate with 25 million.

Also, *frames* of 1024 bits are clocked into a set of FFs and loaded in parallel to a *frame* of latches as the file is loaded.

Also, in some FPGAs, the very long scan chain is actually divided into *multiple smaller chains*, that are loaded by the *configuration port*.

The *configuration port* has several modes, controlled by dedicated pins.

Mode Pins	Mode
0 0	Serial load with FPGA as master
0 1	Serial load with FPGA as slave
1 0	Parallel load with FPGA as master
1 1	Parallel load with FPGA as slave

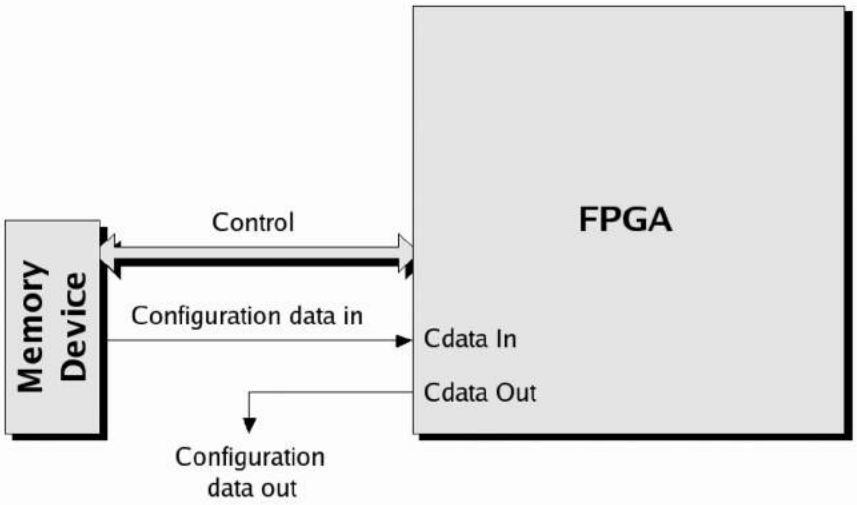
Each vendor defines this differently.

Configuring the FPGA

Other pins are used to tell the FPGA to commence with the configuration, and to report an error and that the configuration is complete.

The pins dedicated to the *configuration port* can be reused as general purpose I/O once the configuration is complete.

The **serial load with FPGA as master** is the simplest mode.



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

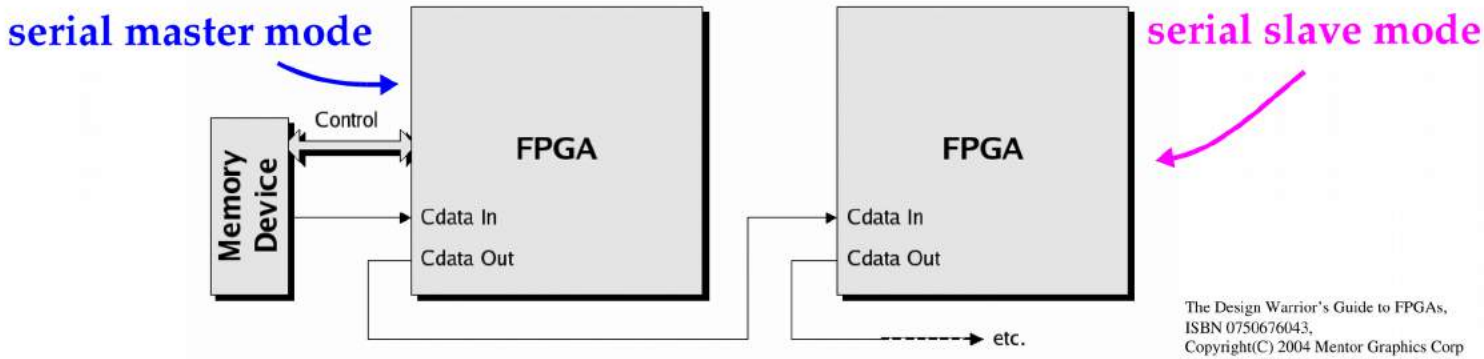
The *memory device* is usually a special FLASH with a single data out pin.

Configuration Modes of the FPGA

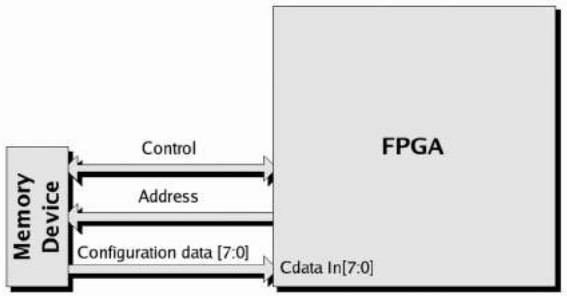
The FPGA controls the FLASH device through a reset and clock pin.

In this mode, there are no addresses that need to be issued by the FPGA.

As indicated, the *Cdata Out* pin is used when daisy-chaining.



In contrast, **Parallel load with FPGA as master** uses addresses.



Configuration Modes of the FPGA

Here, the FPGA maintains an internal counter that generates the byte addresses.

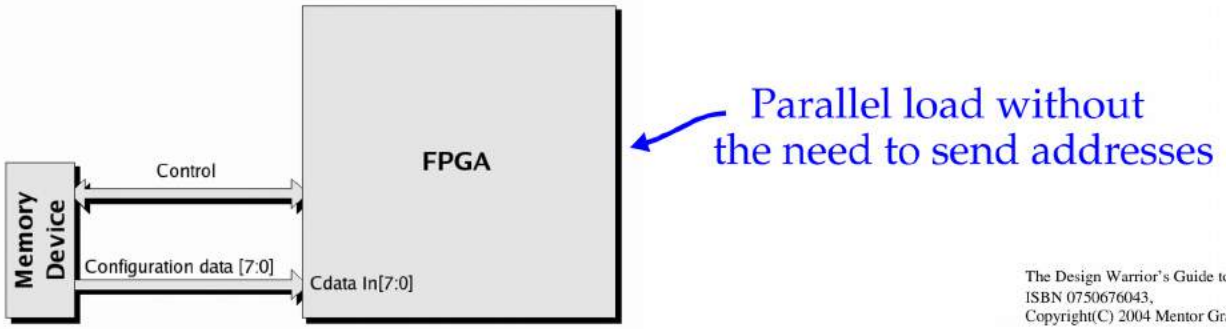
The advantage is reduced configuration time and off-the-shelf memory chips.

However, in the early days, the fetched byte was serially clocked into the scan chain, so it wasn't faster.

Similar to *serial mode*, these pins can be reused as general purpose I/Os.

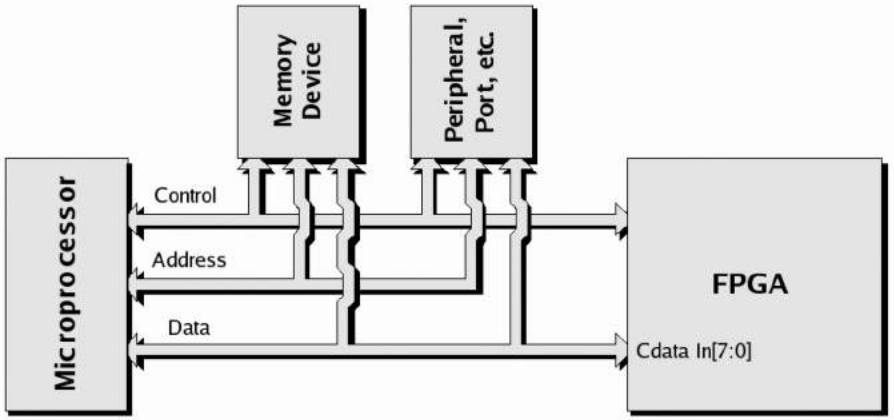
Unfortunately, this is not often used because of the load associated with the hard-wired memory device.

Modern devices use a variation of the *serial load*.



Configuration Modes of the FPGA

Boards that incorporate microprocessors can use the **parallel load with FPGA as slave** mode (or **serial load with FPGA as slave**).



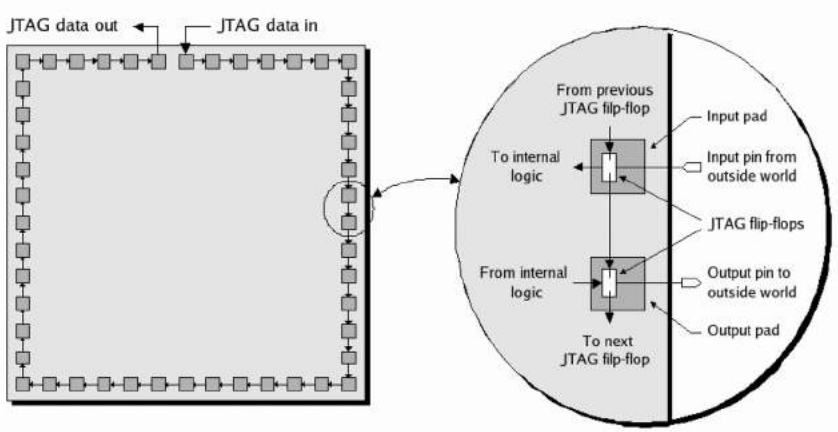
In this case, the microprocessor fetches the byte data and loads the FPGA.
This allows a great deal of flexibility w.r.t. how the chip is programmed.

The **JTAG port** can also be used to program the FPGA.
Joint Test Action Group -> JTAG.

JTAG was originally introduced to enable *boundary scan*, which is used to test PCBs and the chips on the board.

Configuration Modes of the FPGA

The JTAG port has several dedicated pins, two of which are for data input and output, and an internal *scan chain* the connects to all FPGA I/O pins.



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

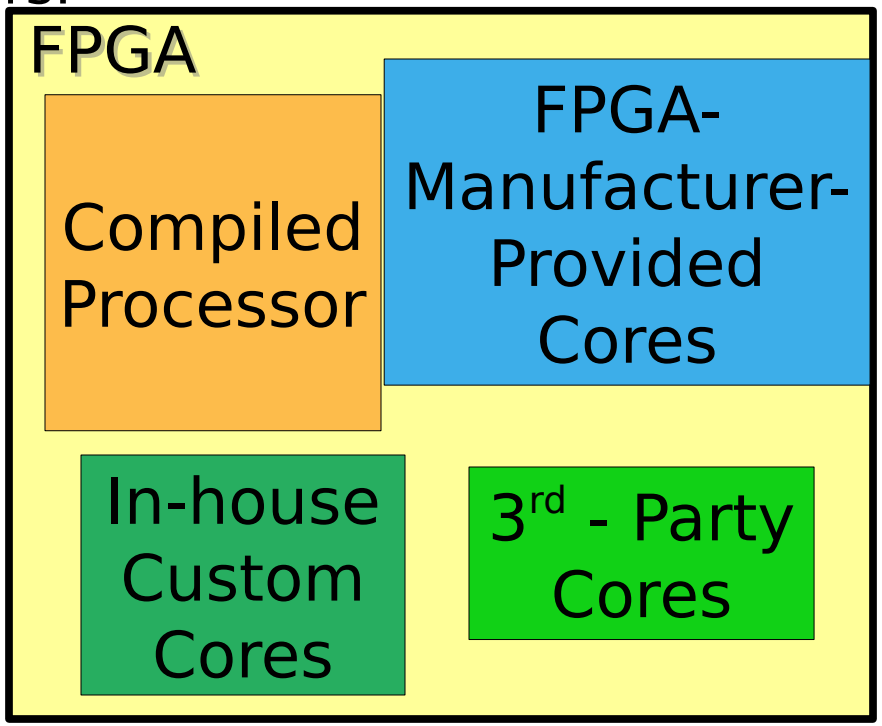
For *boundary scan* testing, data is scanned into the I/O scan registers, the chip operates on the data and the results are scanned out for verification.

The JTAG port also contains a command register (not shown) that can be loaded to inform the FPGA to get its configuration data from the JTAG port.

Thus, typically there are 3 mode pins in today's FPGA to allow this mode to be specified (and another, see text for an embedded processor mode).

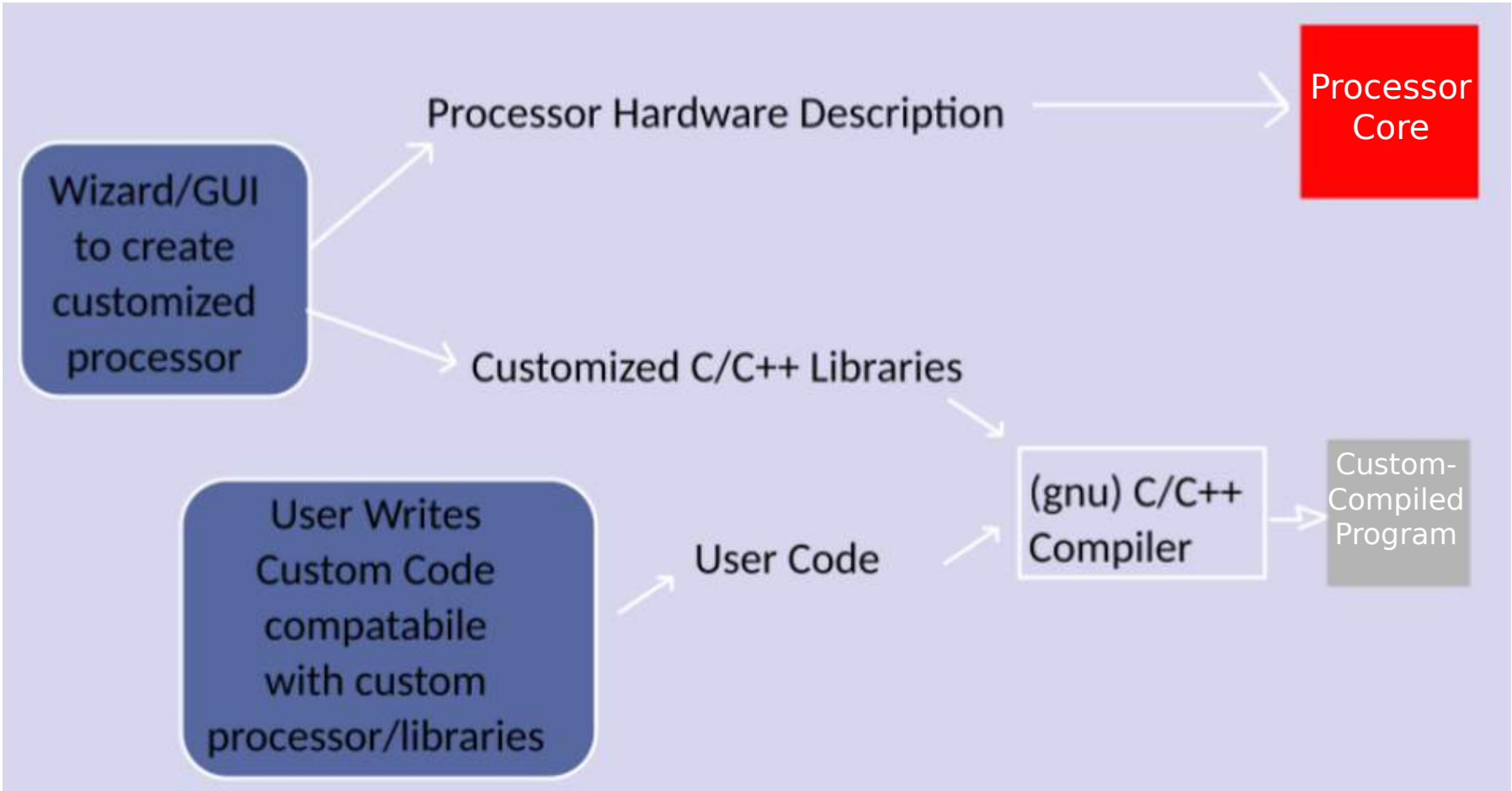
Soft Processor Development

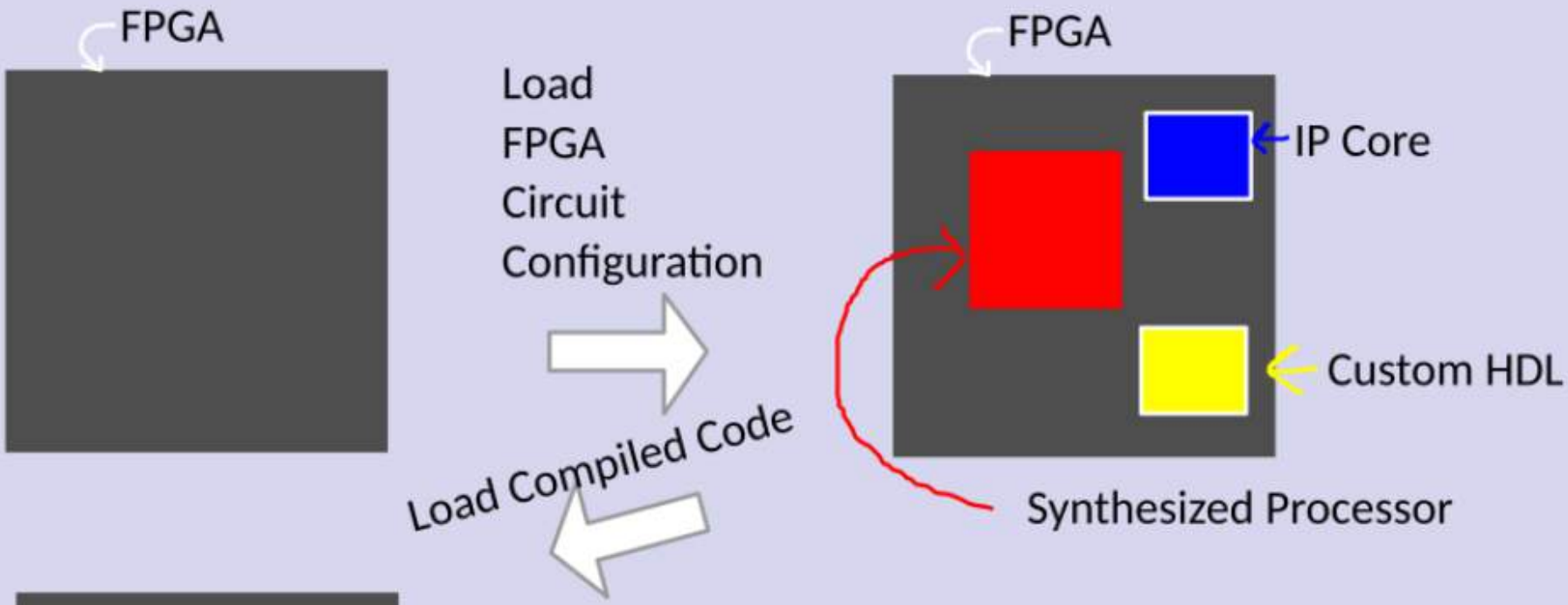
- If sequential code really is appropriate, add a processor.
- A **Soft Processor** is a compiled, customizable processor (not to be confused with fixed processors sometimes provided on FPGAs)
- examples of options and parameters:
  - cache size, #bits,
  - on-FPGA RAM,
  - external RAM controller
  - branch prediction...
- Some offer even more features typical on microcontrollers like
  - SPI (serial peripheral interface)
  - pulse width modulators
  - Custom instructions
    - hardware accelerate frequently executed code



"Soft" Processor Core Development

- A processor needs a code to run so an integrated (code) development environment (IDE) is provided along with



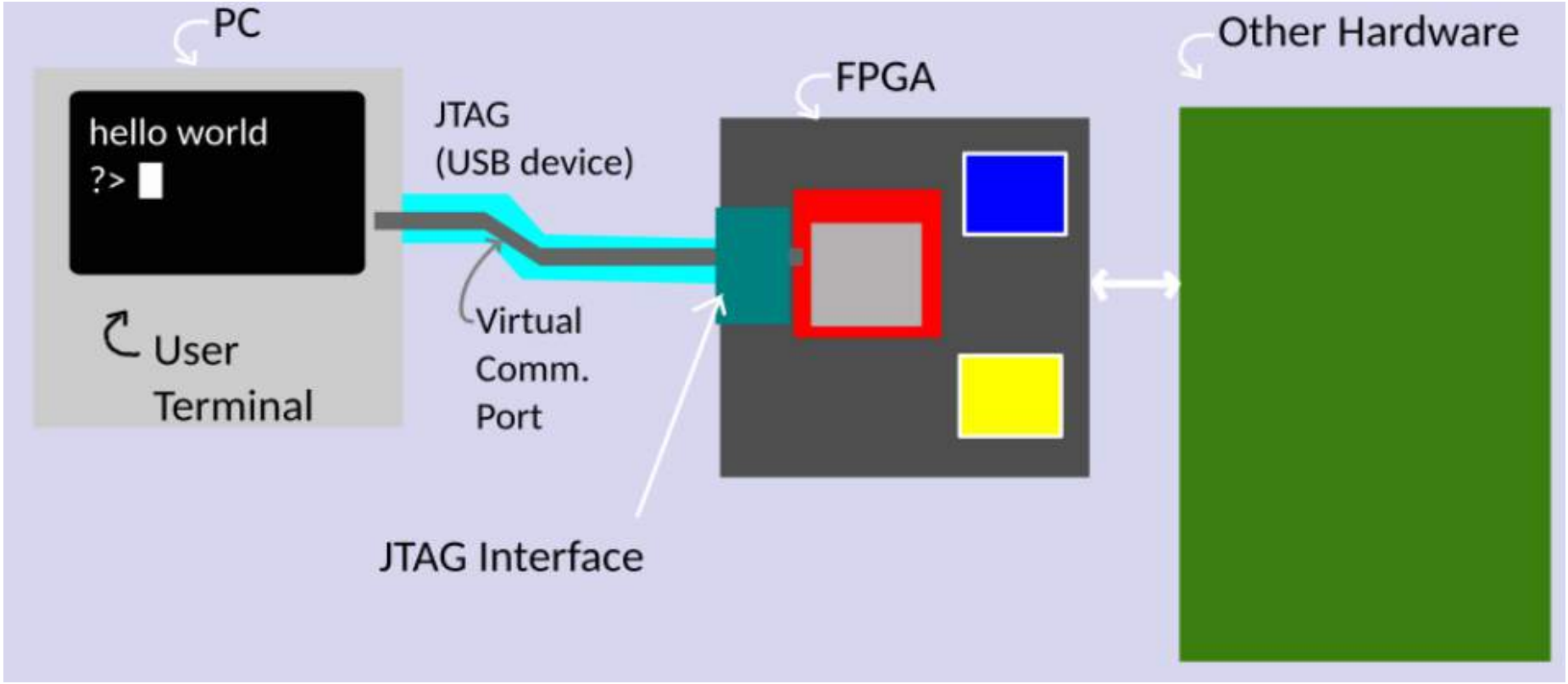


- IDE provides
  - editor, debugging gui,
  - on-chip debugging, variable
  - monitoring and altering and execution control possible
- Advanced features like on-the-fly hardware break-points depend on processor configuration

Xilinx Vivado Design Suite

- Whereas the previous approaches considered a processor as one component in a design, modern approaches view the processor as the central element and additional logic as an addon
- Emphasiszes SOC,
  - Processor and IP-Centric Design (maximal using pre-built cores)
- and
- Mixed development of C/C++ (soft procesors) and other IP-based design

A FPGA-based System



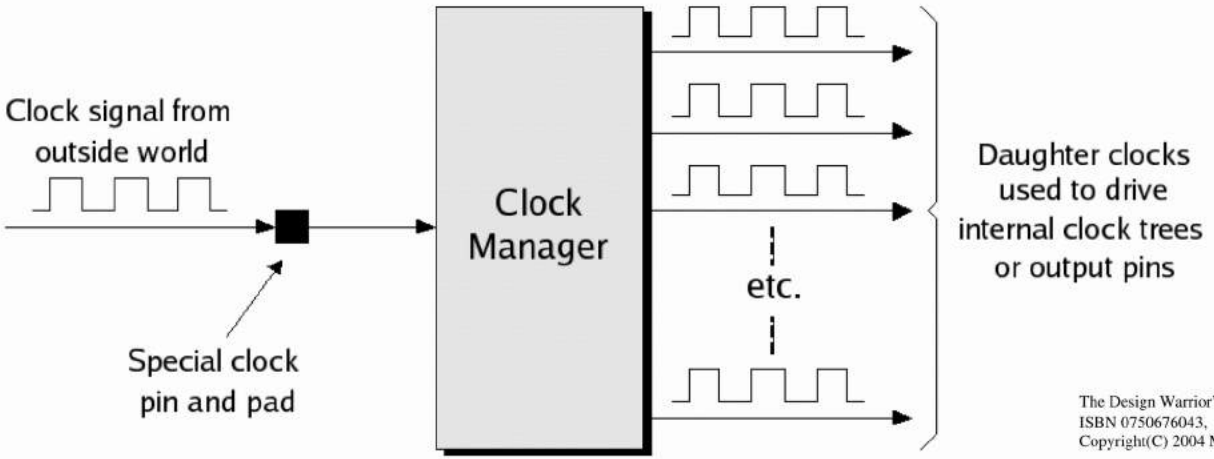
- Most basic user interface is a terminal hosted on PC
- Interaction through Matlab, a Labview Gui, etc.. is common too. USB, USB-JTAG, and LAN would be the most typical conduits

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**.

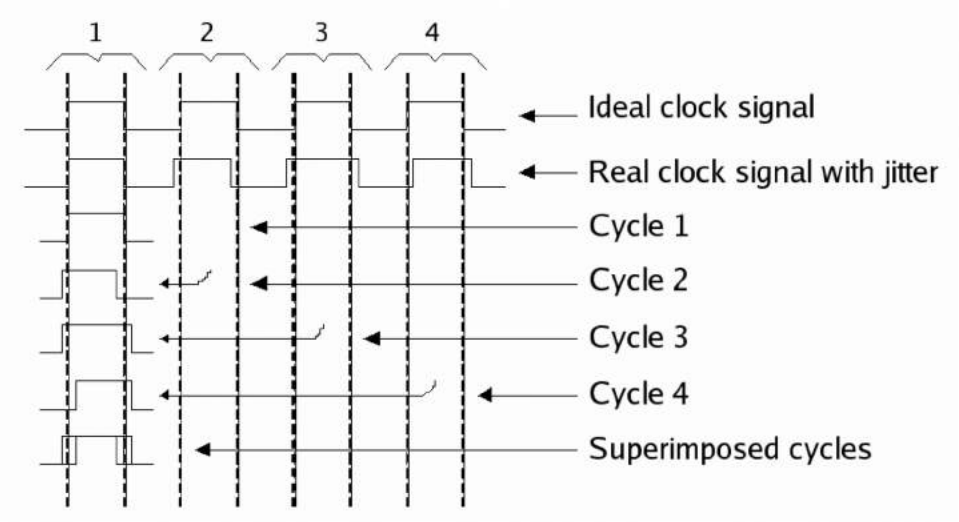


Clock Trees and Clock Managers

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:

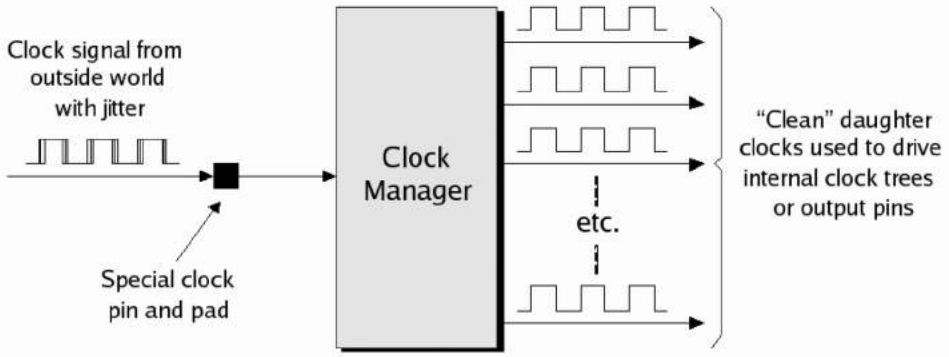
- *Jitter removal*: Uncertainty in the exact arrival time of each clock edge results in jitter.



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

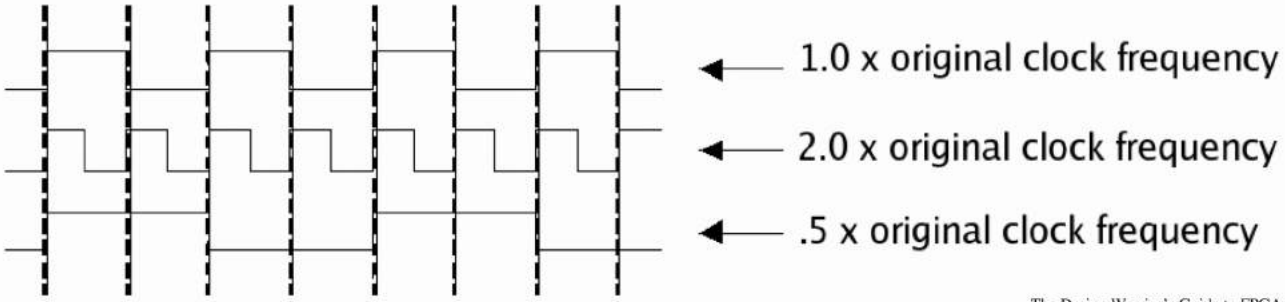
Clock Trees and Clock Managers

- *Jitter removal*



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

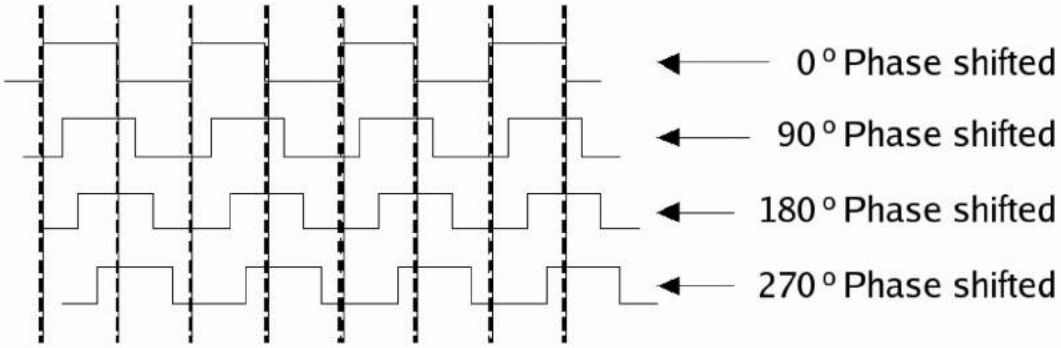
- *Frequency synthesis*: Allows the external clock frequency to divided or multiplied.



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.



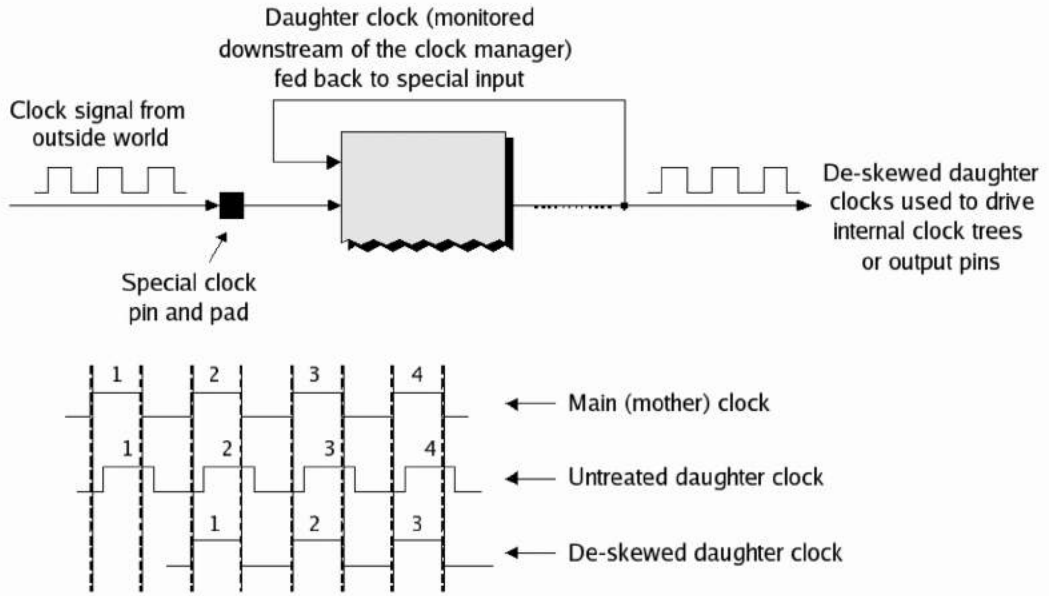
Common phase shifts are 120 and 240 degrees (for 3-phase clk schemes) and 90, 180 and 270 degrees for 4-phase schemes.

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.

Sea of Gates

- FGPAs are programmed with software, though we want to think of it as hardware
- HDLs are software programming languages whose functionality is mapped to a hardware implementation
  - don’t think of the FPGA as “running” the code or as a simulator
  - want to consider the hardware the HDL code maps to
- At a high-level, an FPGA is like a “sea of gates” that we wire up but there are nuances that make some functionality descriptions map to better implementations
- Nuances to the “sea of gates” generalization include
  - Dedicated carry-chain support between blocks, dedicated hardware Multipliers and DSP Slices and other HARD IP CORES, Block RAM vs Distributed (LUT) RAM
  - Understanding of limited dedicated clock routing and clock domains
  - Increasing role of soft and hard processors in FPGA design
  - Understanding of additional FPGA features like high-speed IO