UMBC

AN HONORS UNIVERSITY IN MARYLAND

Department of Computer Science and Electrical Engineering

CMPE 415 Programmable Logic Devices

FPGA Technology III

Prof. Ryan Robucci

Some slides (blue) developed by Jim Plusquellic

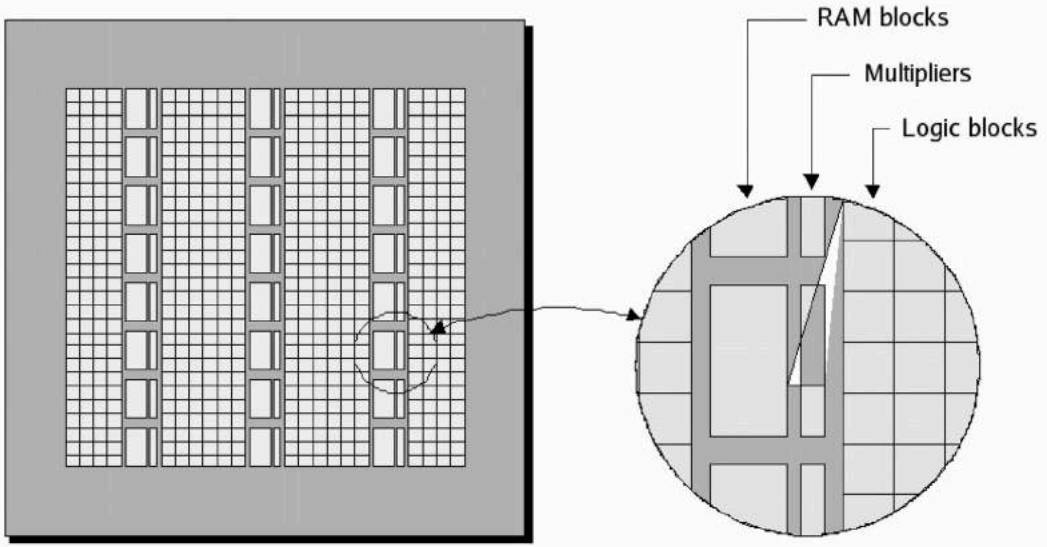
Some images credited to book: Maxfield, Clive. The design warrior's guide to FPGAs: devices, tools and flows. Elsevier, 2004.

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.

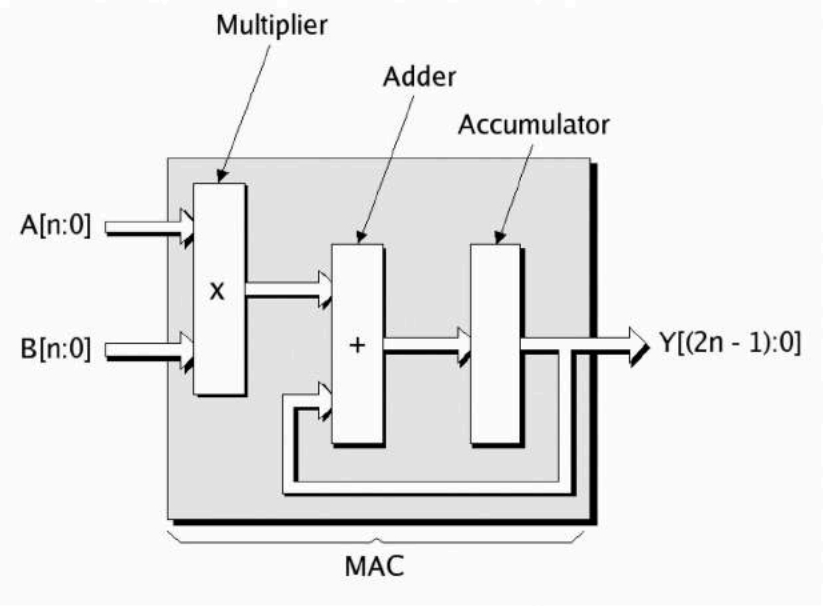
They are often located near the *eRAM* blocks.



FPGA Additional Features and Characteristics

Some FPGAs offer dedicated adder blocks.

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



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.

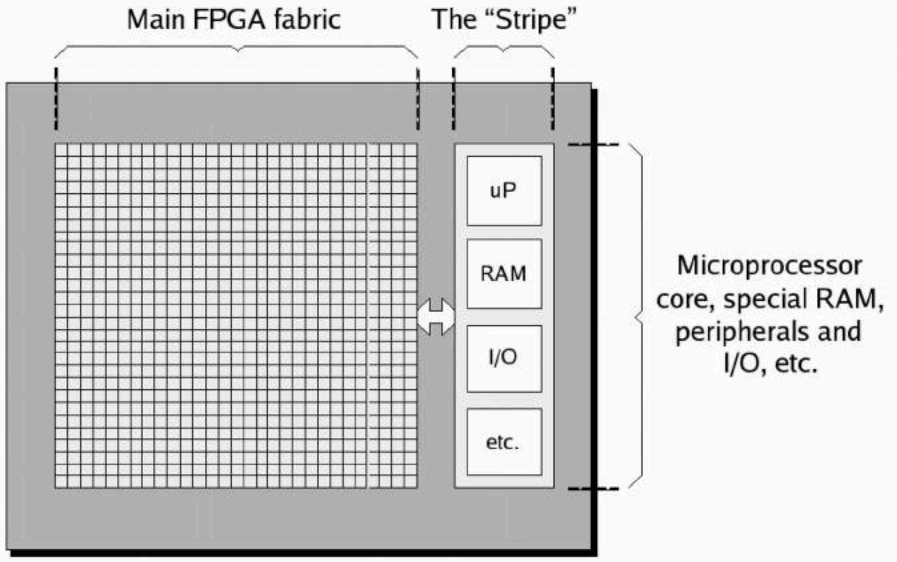
Until recently, these appeared as discrete chips on the PCB -- now they appear as embedded **microprocessor cores** within the FPGA.

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.



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)

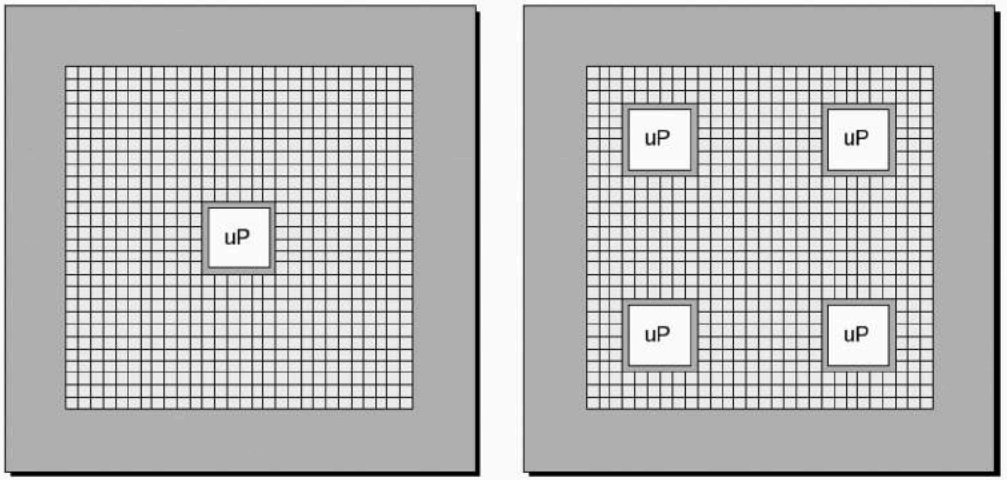
- In a strip (called the Stripe) to the side of the FPGA fabric (cont).

  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)

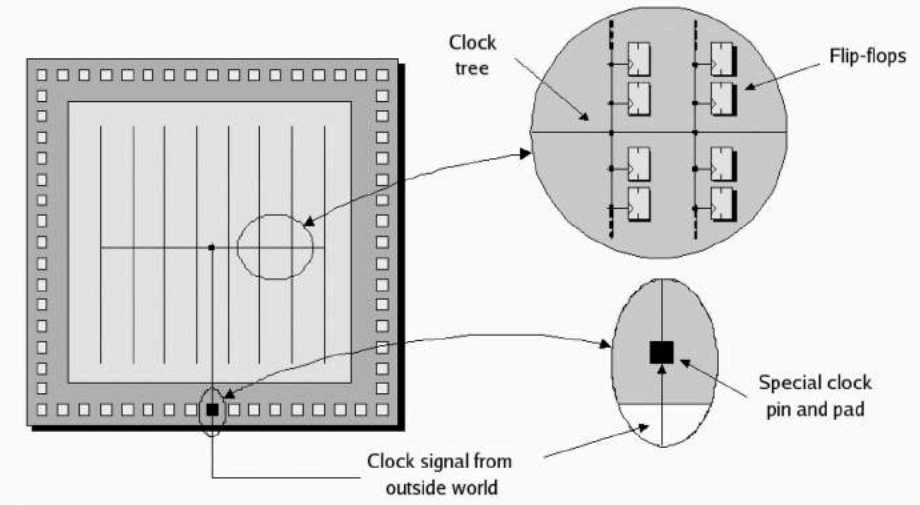
For *soft microprocessor cores*, a group of PLBs are configured as the micro.

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

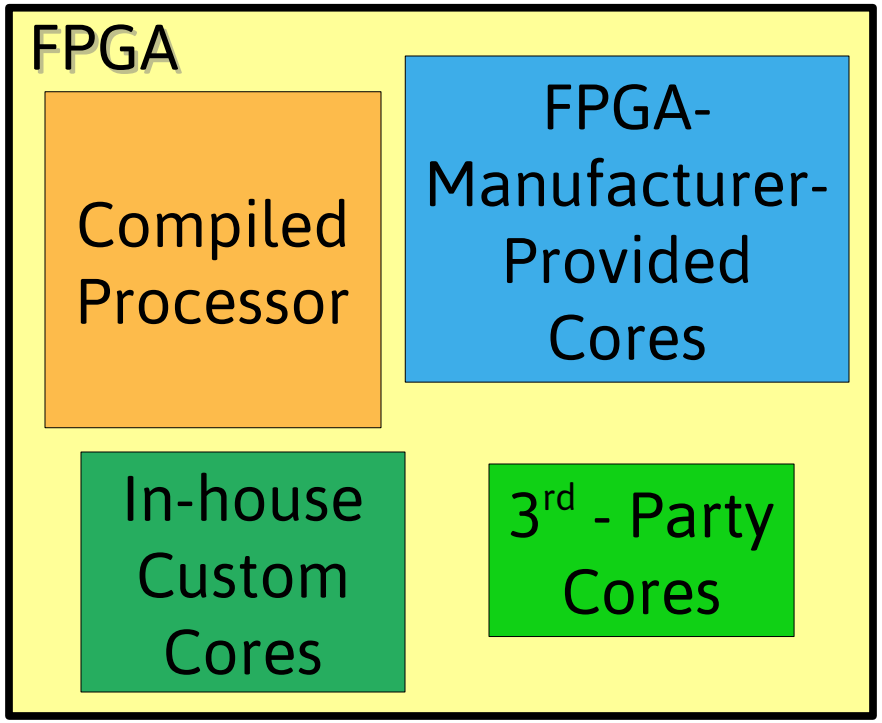
All synchronous FFs in the chip are driven by an external clock signal.



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

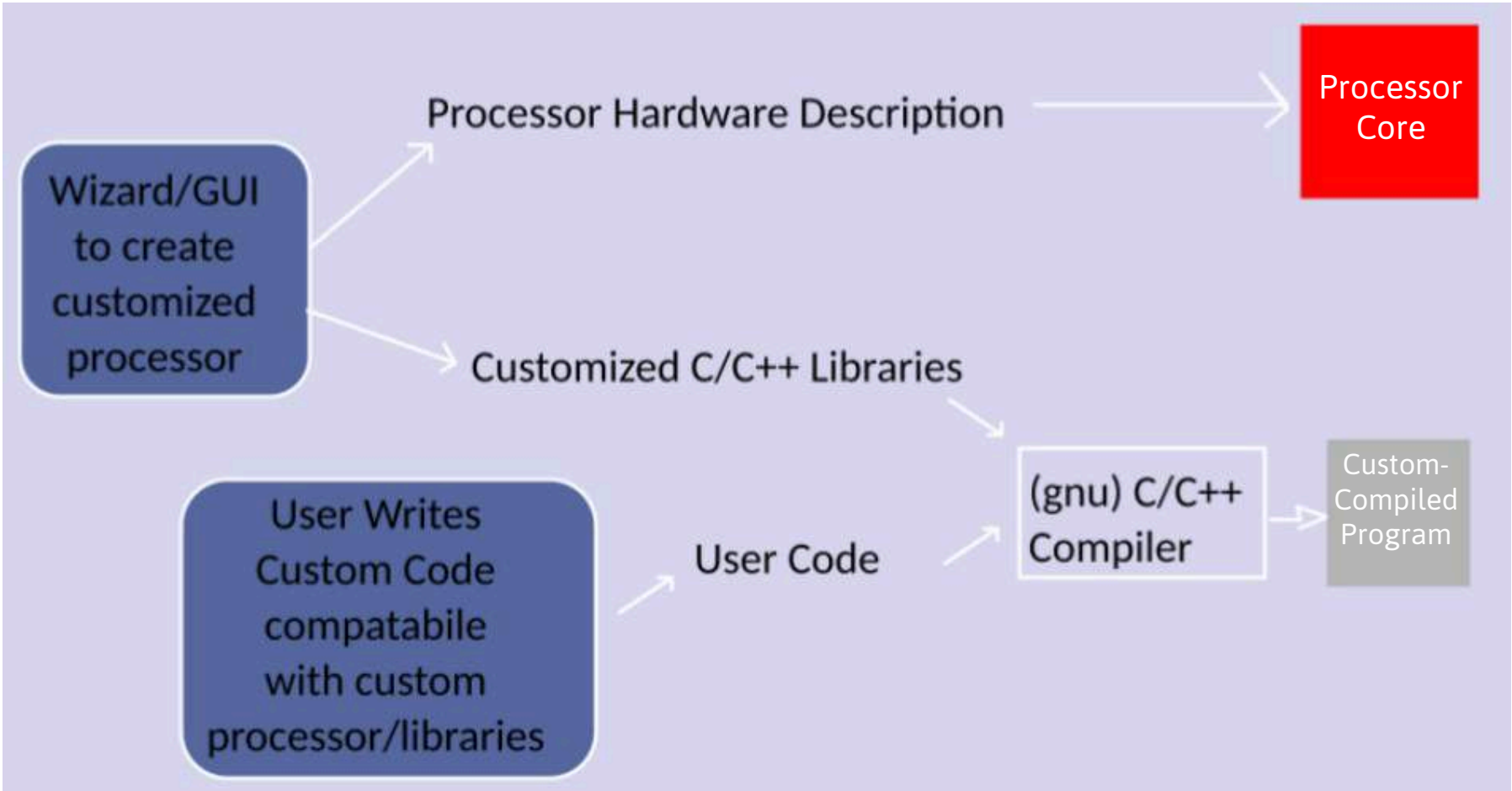
Soft Processor Development

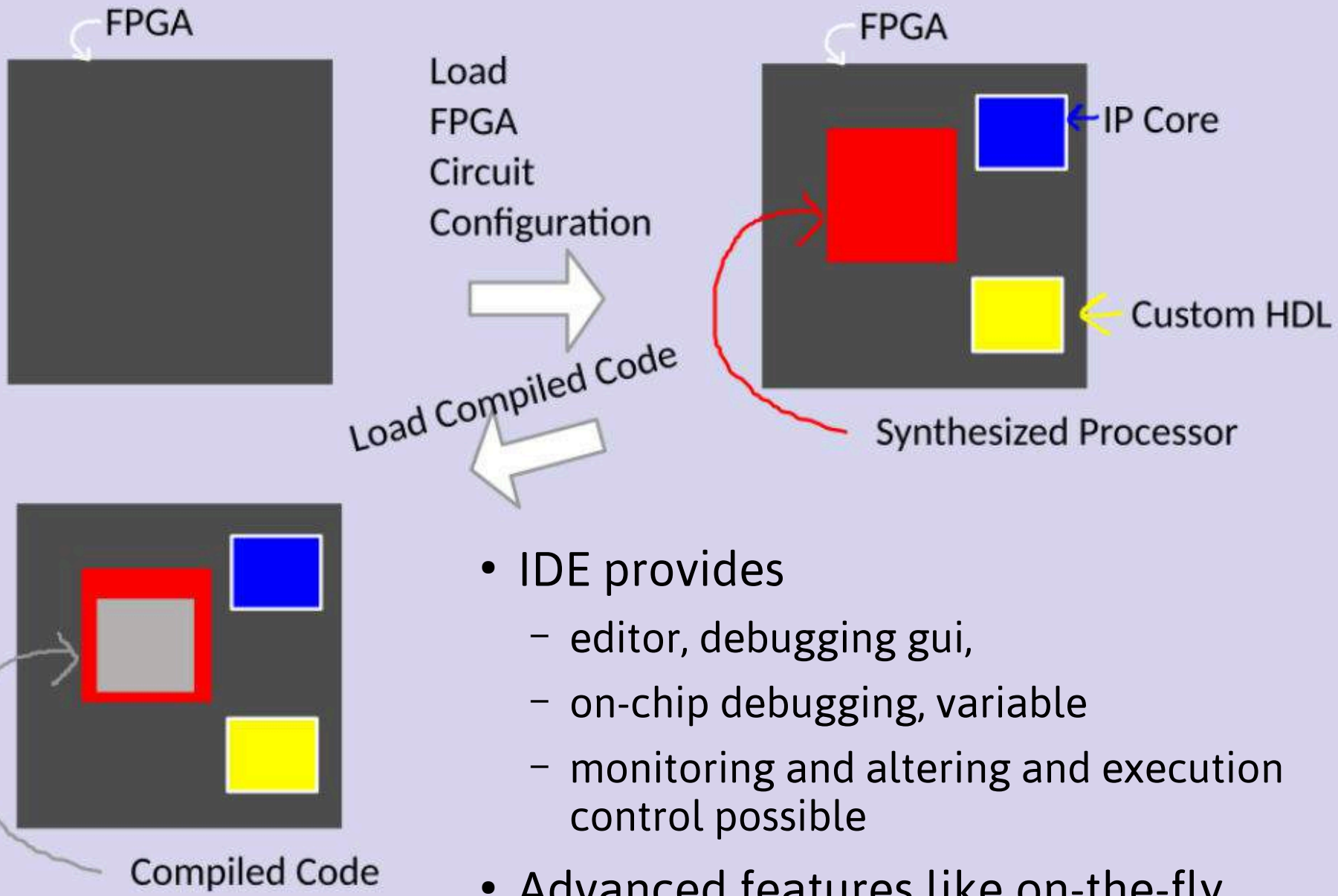
- If sequential code really is appropriate, add a processor.
- A **Soft Processor** is a compiled, customizable processor (not to be confused with fixed processors sometimes provided on FPGAs)
- examples of options and parameters:
  - cache size, #bits,
  - on-FPGA RAM,
  - external RAM controller
  - 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 code to execute, so an integrated (code) development environment (IDE) is provided along with the FPGA tools.



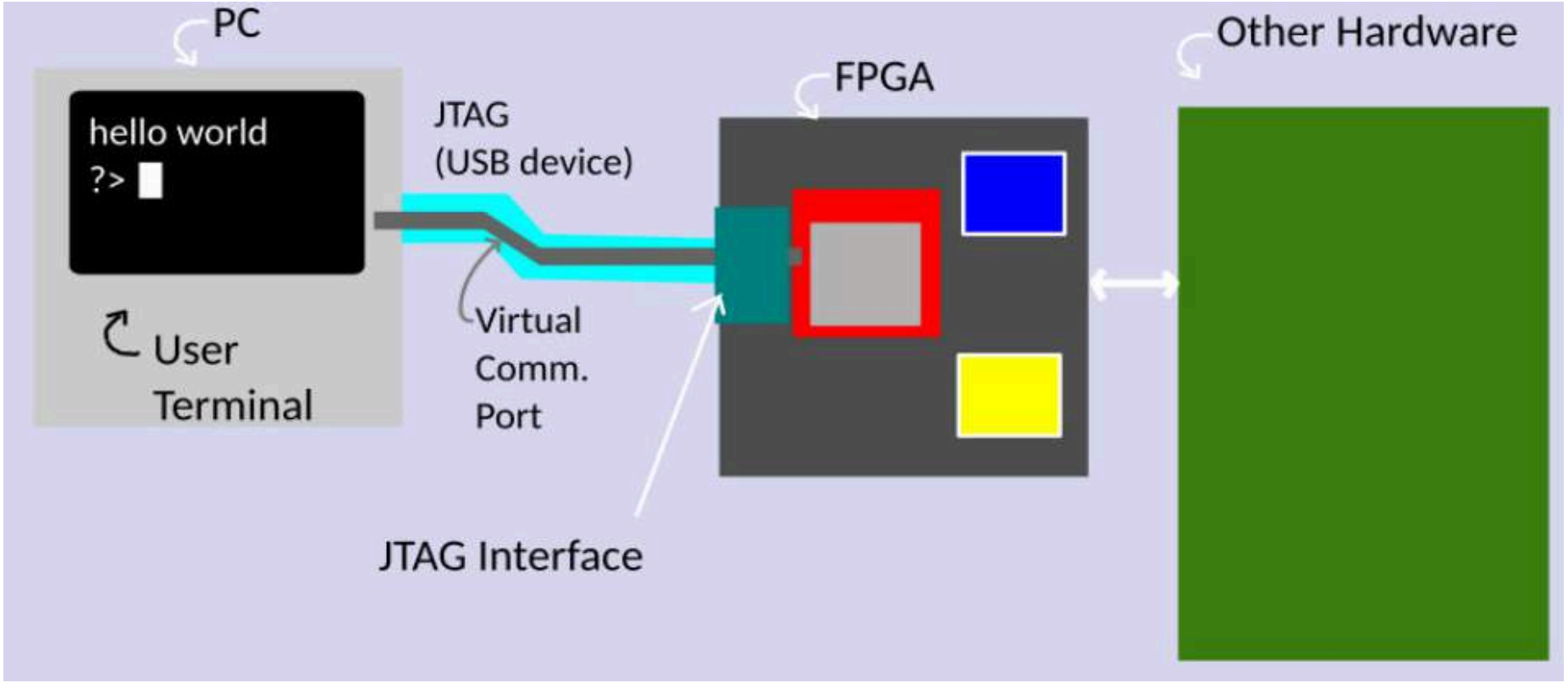


- IDE provides
  - editor, debugging gui,
  - on-chip debugging, variable
  - monitoring and altering and execution control possible
- Advanced features like on-the-fly hardware break-points depend on processor configuration

Processor and IP-Core Centric Design

- Whereas the previous approaches considered a processor as one component in a design, modern FPGA Tool approaches (e.g. Xilinx Vivado Design Suite) view the processor as the central element and additional logic as an add-on
- Emphasiszes SOC
  - Processor and IP-Core-Centric Design (maximal using pre-built cores)
- Mixed development of C/C++ (soft procesors) and other IP-Core-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

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*.

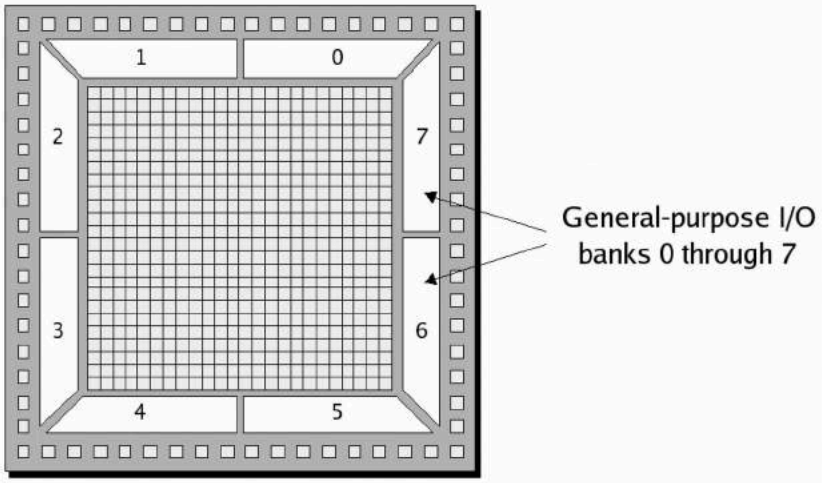
They can be configured to accept and generate signals conforming to any one of these standards.

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.

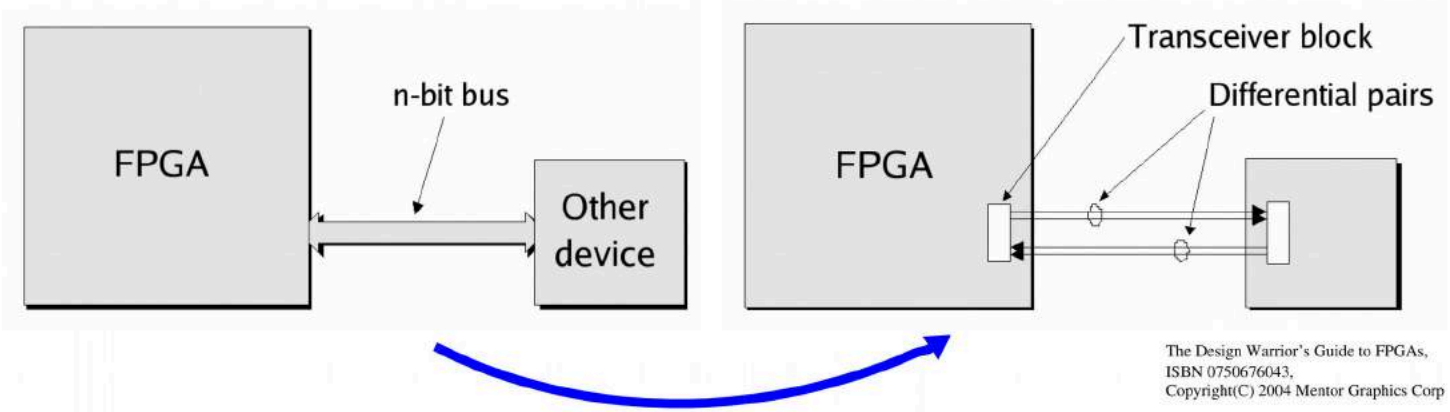
In order to accommodate the variety of voltages, each I/O bank has its own supply pins, that are separate from the *core* supply pins.

Off-Chip Communication
High-Speed Serial vs. Parallel and Serialization

- High speed communication requires
  - fast signal transitions which create reflections within a wire and interference with other wires
    - Mitigation requires controlled impedance which means the lines must have very consistent physical manufacturing to ensure 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?
  - Cabling becomes difficult and expensive for high-speed parallel,
    - it is difficult to create a 64 bit bus with matched electrical characteristics
  - At some point it becomes necessary to design for high speed signals. Past that point, it can be easier to work with fewer signal lines.
- **Gigibit Transceivers** support **serialization** of parallel-bit data words to a high-speed serial data-stream, and **de-serialization** to reconstruct the parallel bits in a lower-freqeuncy clock domain

Gigabit Transceivers

FPGAs started including *hard-wired gigabit transcievers*.



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.

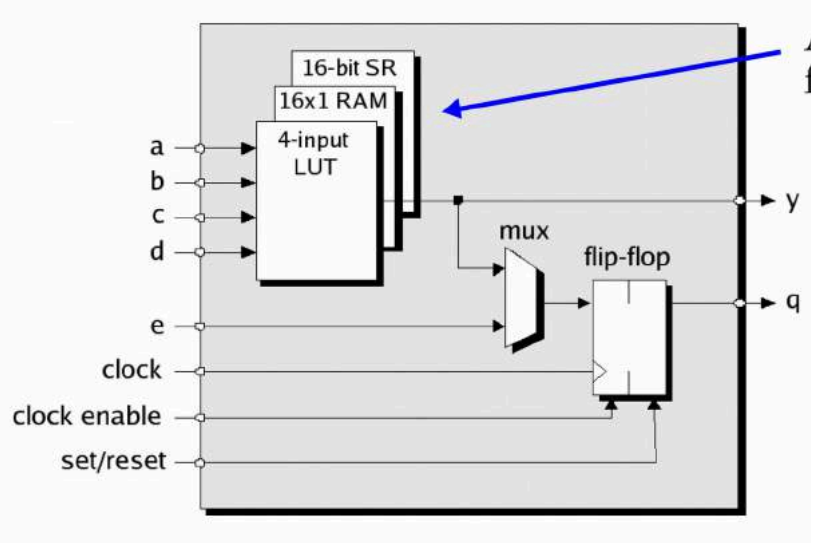
Design Complexity vs Size

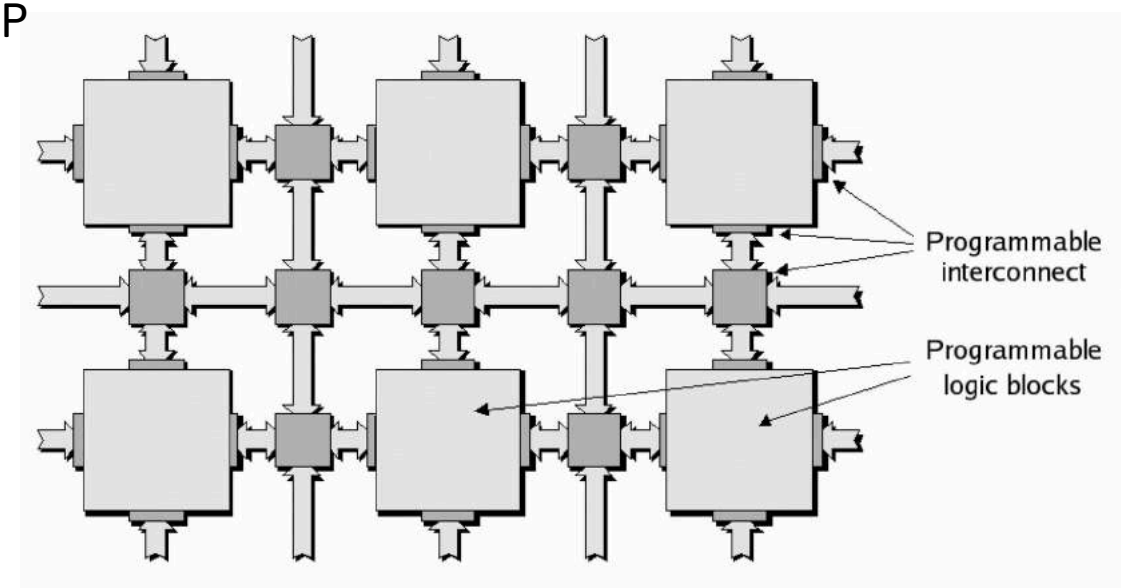
- A regular processes early in ASIC physical design is to estimate the size based on "design complexity"
- In ASICs, "design complexity" is mainly quantified by a design's "**gate count**" which is used to estimate the required chip size
  - A gate is not well-defined, but but for the purpose of the estimate it could be defined as a 2-input nand gate equivalent
  - The complexity could also be measured by "transistor count" and divided by 4 to generate an equivalent nand-gate count
  - The combinatorial circuit could be distinguished from the sequential elements resulting in an estimate based on combinatorial "gate count" and sequential "register count"
- FPGAs don't actually directly implement gates.
  - Instead they provide another manufacturer specific functional blocks (e.g. logic cells/logic blocks/slices) and provide an average equivalent **gate count** to help estimate the supported design complexity
- This feature measure (e.g. # of logic cells) along with the other specialized features listed on a product sheet (help consumers understand the comparative size and capabilities of the various FPGAs sold by that manufacturer
  - FPGA features to consider when estimating design "size":
    - # of logic cells/logic elements/things equiv. to 4-input LUTs+LATCHES+ETC
    - Count and Size of
      - embedded RAM blocks (block RAM)
      - embedded multipliers – sometimes marketed as DSP blocks/slices or multiply-accumulate (MAC) blocks
      - dedicated embedded adders (accumulators)
    - Number of any other special hardware that represents a significant equivalent gate count for your design

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.

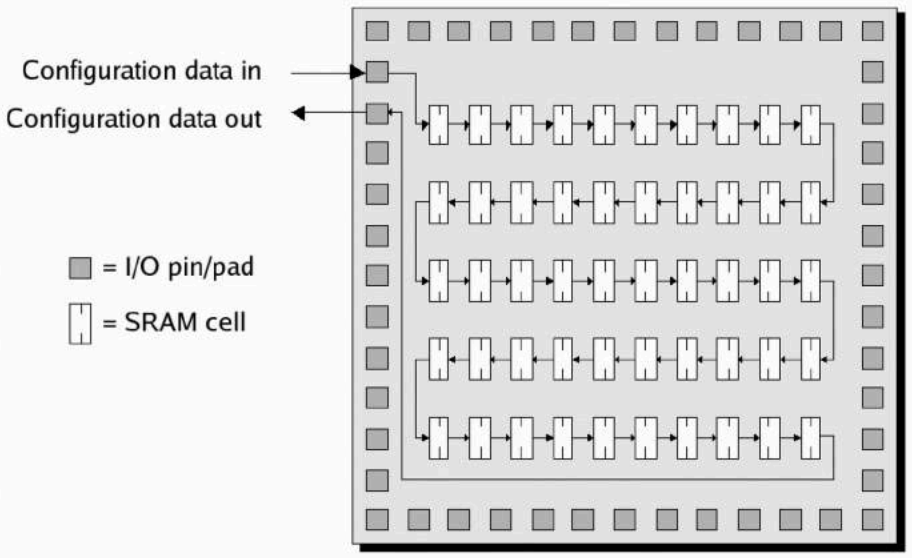
For example, within the PLB, the MUX's *select input* needs a configuration cell.

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

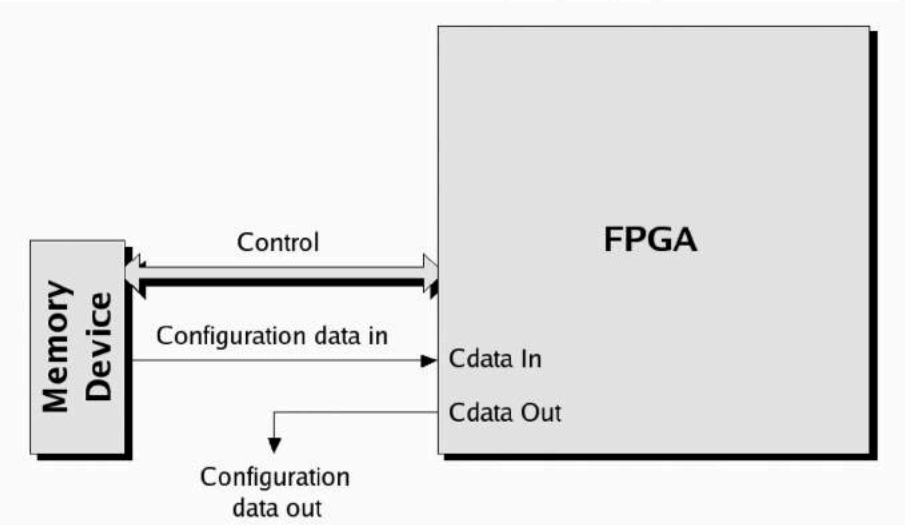
Each vendor defines this differently.

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

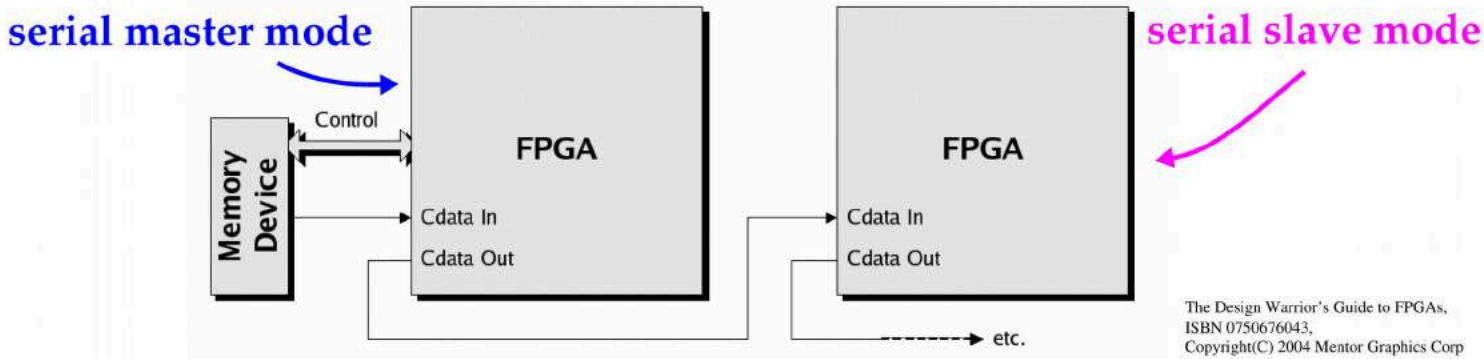
The *memory device* is usually a special FLASH with a single data out pin.

Configuration Modes of the FPGA

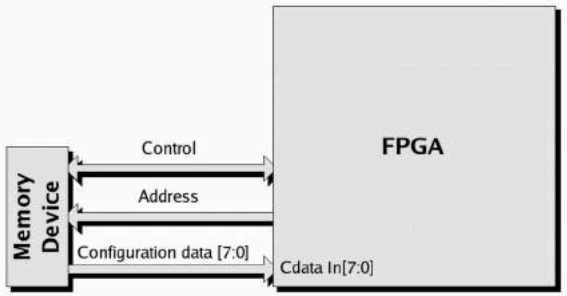
The FPGA controls the FLASH device through a reset and clock pin.

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.

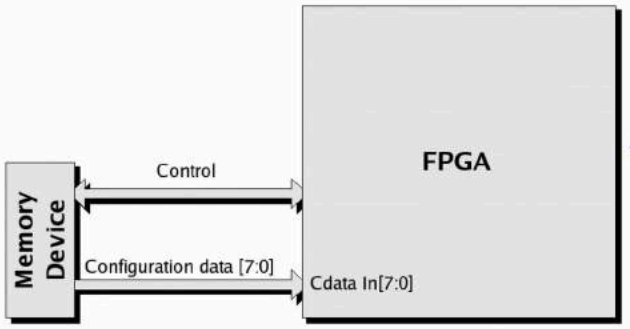
The advantage is reduced configuration time and off-the-shelf memory chips.

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.

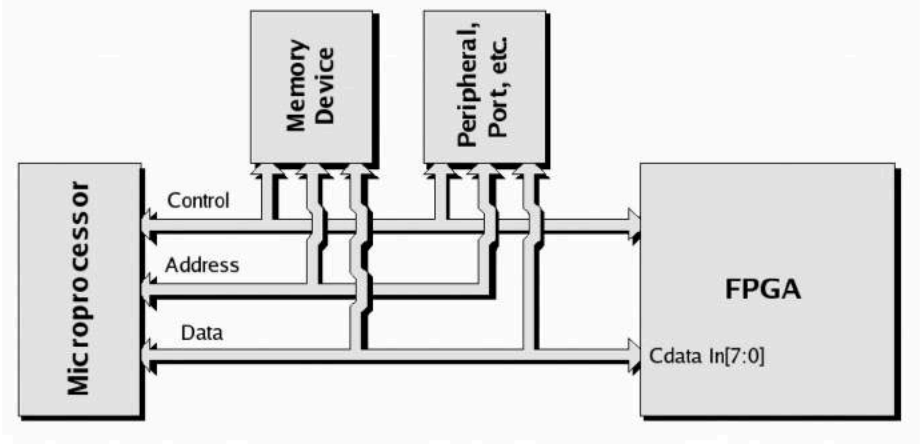
Modern devices use a variation of the *serial load*.



Parallel load without the need to send addresses

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**).



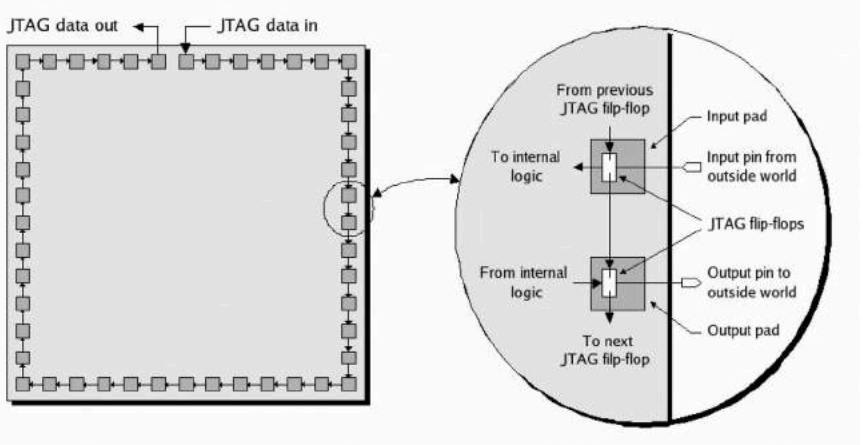
In this case, the microprocessor fetches the byte data and loads the FPGA.
This allows a great deal of flexibility w.r.t. how the chip is programmed.

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

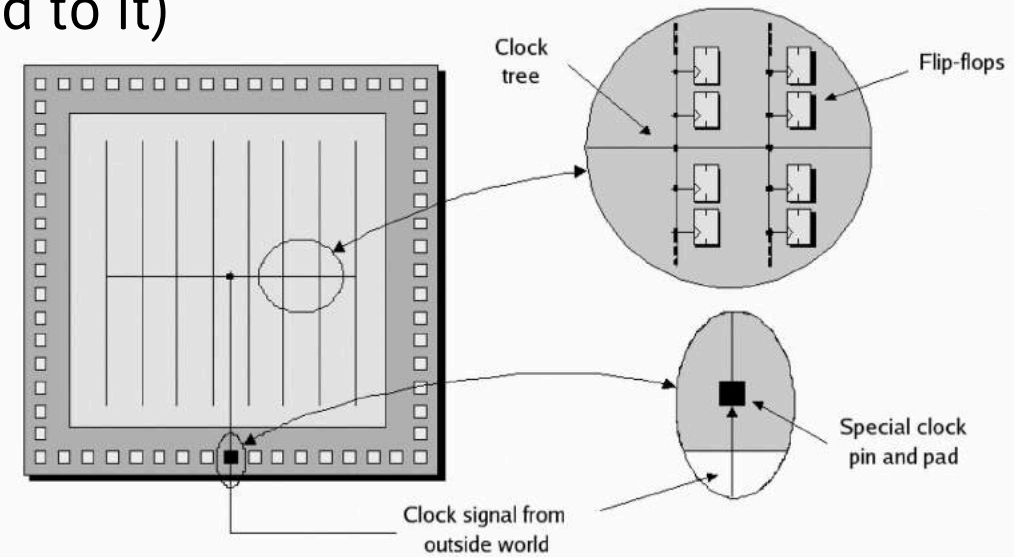
For *boundary scan* testing, data is scanned into the I/O scan registers, the chip operates on the data and the results are scanned out for verification.

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).

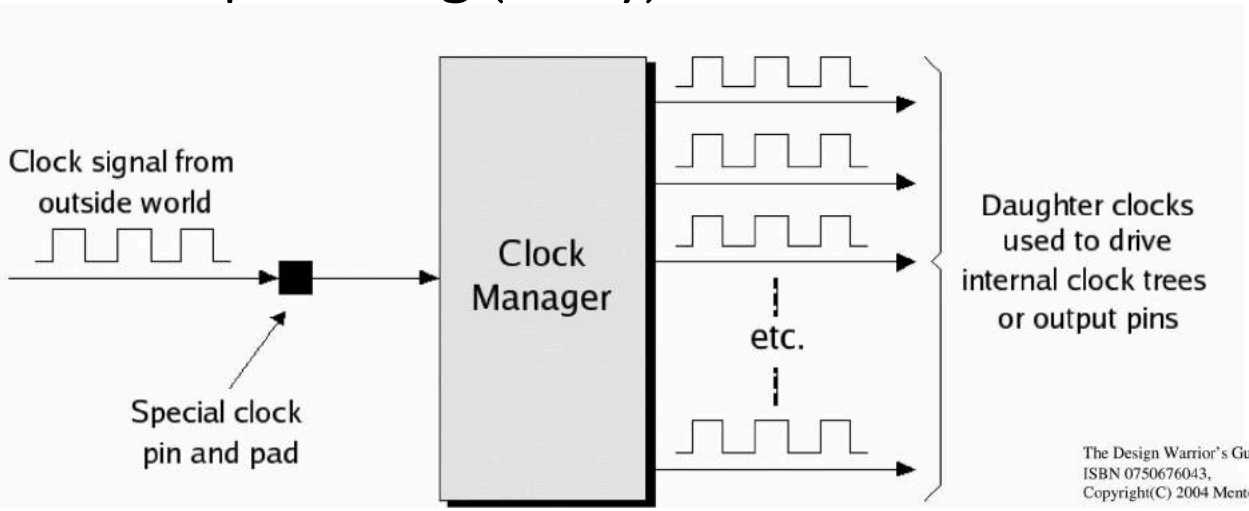
Clock Trees and Clock Managers (I)

- Flip-Flop (FF) clock inputs are driven by special dedicated buffering circuity and routing lines for clocks with the primary purpose to ensure minimal **clock skew** within a **clock domain**
  - Clock skew is the difference in clock edge arrival times at different endpoints (Ffs)
  - A **clock domain** is characterized by the set of FFs connected to a given clock signal (and the associated combinatorial circuity connected to it)



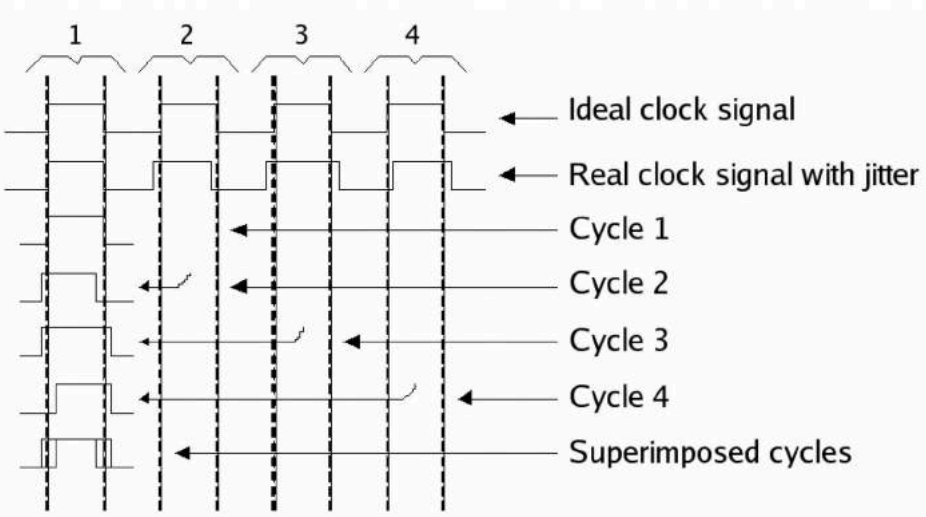
Clock Trees and Clock Managers (II)

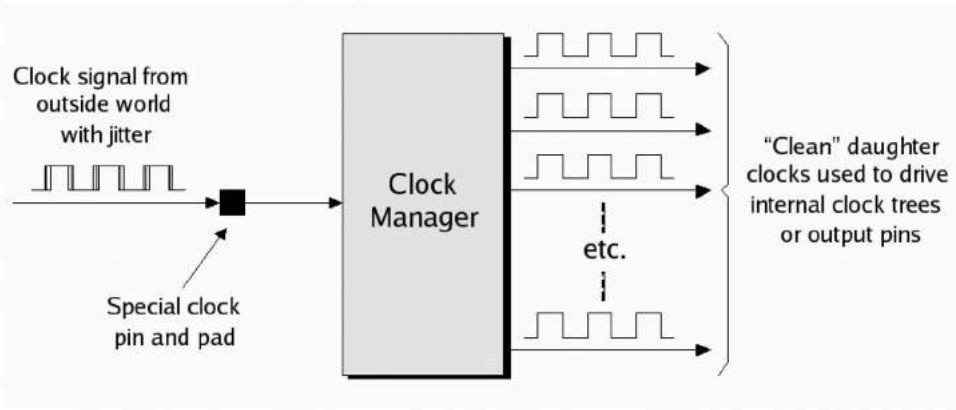
- Several pins with special clock circuity are designated per chip (designated as “global” clock input pins) as best for clock input
  - Multiple clock input pins can be used and multiple clock domains can exist
- Alternatively, many **daughter clocks** might be generated from one master clock using a **clock manager** creating multiple clock domains
  - Each generated **daughter clock** might have a different configurable rate and phase lag (delay)



Clock Trees and Clock Managers (III)

- Daughter clocks will drive different clock trees or even off-chip (board-level or system-level) circuity through buffers located near output pins
- Clock managers can “clean” input clock signals by with **jitter removal,** which is defined as the variation in inter arrival times of clock edges

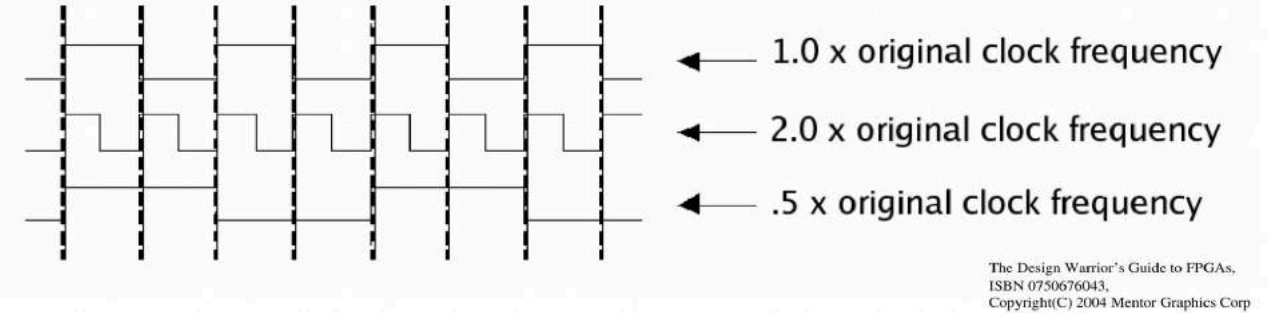




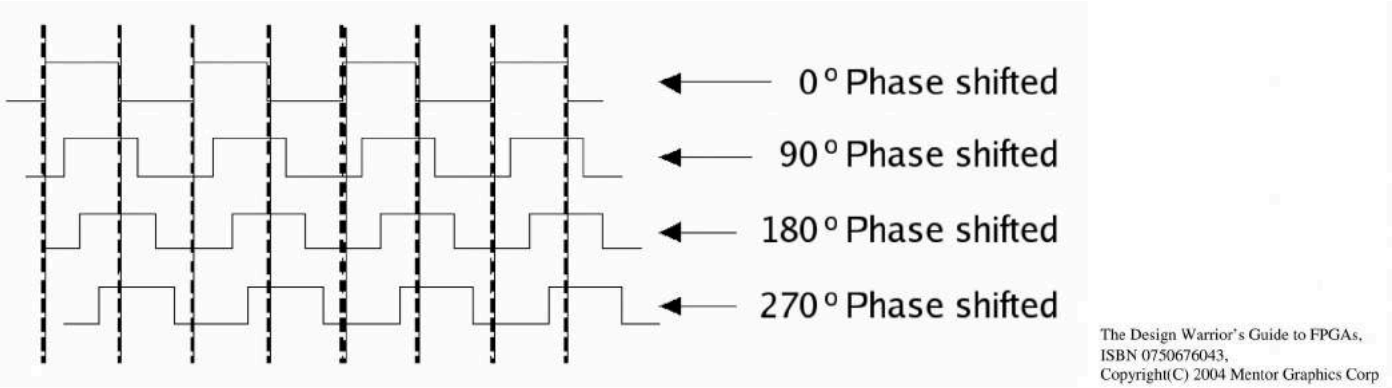
The Design Warrior’s Guide to FPGAs,
ISBN 0750676043,
Copyright(C) 2004 Mentor Graphics Corp

Clock Trees and Clock Managers (IV)

- **Frequency Synthesis** is the capability of clock manager to create daughter clocks with frequencies that are rational fractions (eg 1x ,2 x,1/2 x, 4/5 x) of the master input clock frequency

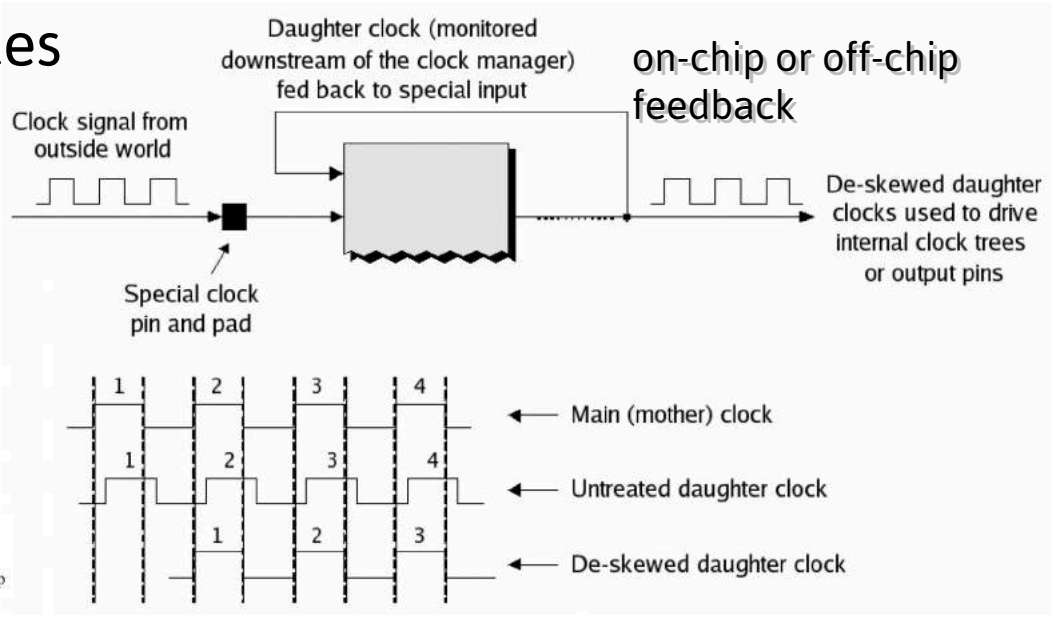


- **Phase shifting** is the ability to generate clocks with defined relative delays (inverted (180 degrees) and 90 degree offsets are common)



Clock Trees and Clock Managers (V)

- **Autoskew Correction** the ability to detect and adjust individual daughter clock delays on-line using feedback and measurement of the clock near an endpoint. In particular use of off-chip board-level feedback accounts for unknown board loads and is used for high-speed memory interfaces



- "Analog" Phase Locked Loops (PLLs) and "Digital" Delay-Locked Loops are two circuit variants for frequency synthesis and phase/skew adjustment. The difference is outside the scope of this lecture and involves characteristics of precision, stability, and noise.

Sea of Gates with Nuances

- FGPAs are programmed with software, though we want to think of it as hardware for most design purposes in this class
- 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