



AN HONORS UNIVERSITY IN MARYLAND

Department of Computer Science and Electrical Engineering

CMPE 415

Programmable Logic Devices

Introduction

Prof. Ryan Robucci

What are FPGAs?

- Field programmable Gate Array
 - Typically re programmable as opposed to one-time programmable.
 - In-system programmable (ISP)
- Can implement
 - Misc. or "glue" logic between systems
 - Full ASIC replacement
 - DSP - via custom logic and/or in-built DSP cores
 - Microcontrollers - implemented as a customizable soft processor
 - High-speed communications (physical layer)
 - Reconfigurable computing (adapting hardware-accelerated behaviour)

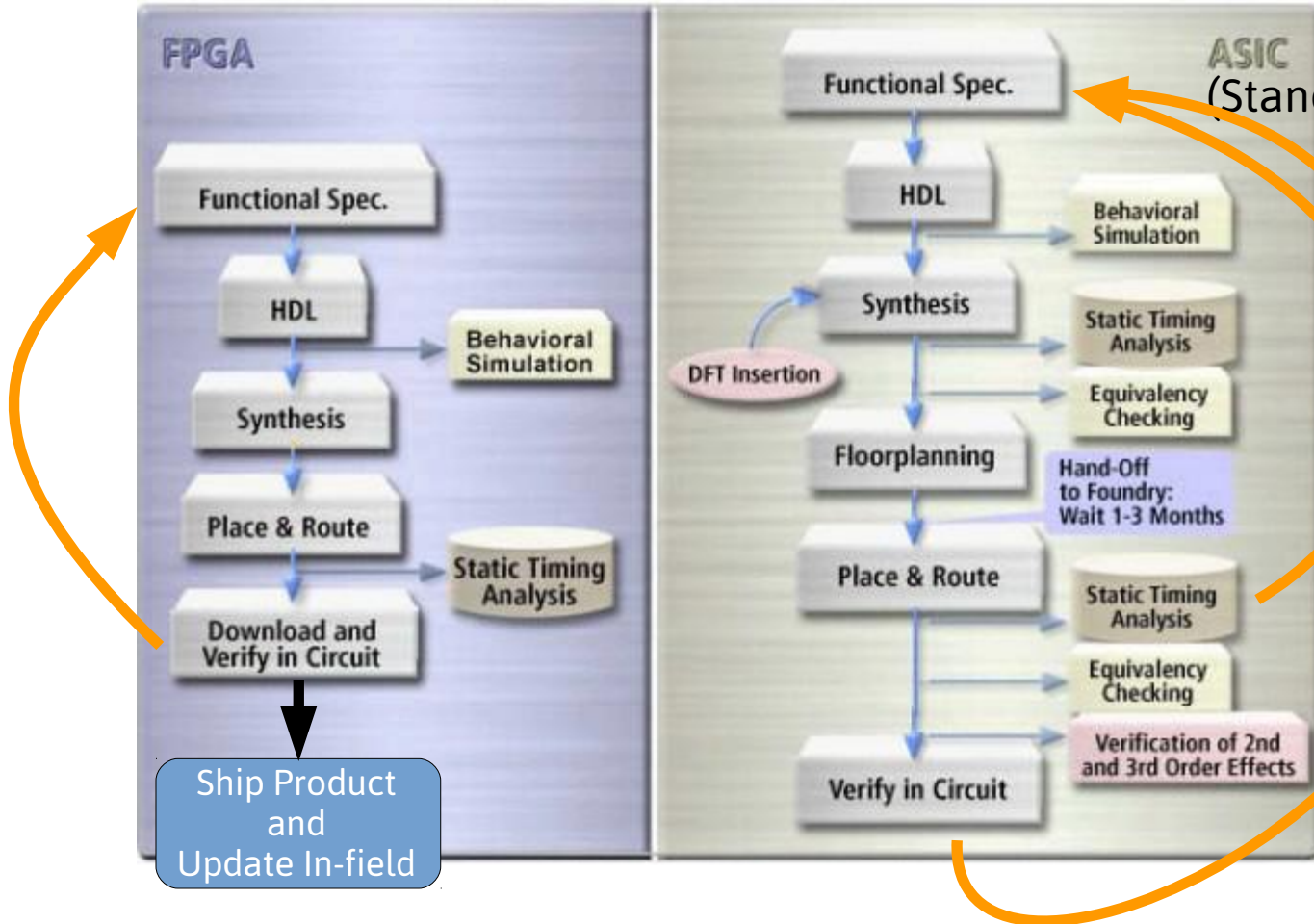
FPGA

- **Field**
 - **In-field programming** means the chip the intention is for it to be reprogrammed even after being placed on board
 - It does not require complex programming hardware such that it can only be done at a factory
- **Programmable**
 - After physical design and manufacturing, it can be reconfigured to perform many functions and satisfy needs for many applications
 - As opposed to an ASIC (application specific integrated circuit) a custom ic designed for one application/function
 - A distinction between **one-time programmable** and **reprogrammable** should be understood. Almost all FPGAs are reprogrammable since there configuration is loaded and stored in SRAM cells. Other devices may use fuse technology that can be “burned” once and thus are one-time programmable. Some devices may use “flash” type memory design to be programmable mutiple time but not as many as a SRAM device.
- **Gate Arrays**
 - Arrays of
 - and,or,xor,nand, etc...
 - Muxes, 1-bit adders
 - registers,latches
 - LUTs (lookup tables)
 - DSPs, PowerPCs
 - GB serializers,etc...

ASIC vs FPGA Design

- FPGAs have a faster design process, but as a faster design cycles

<http://www.xilinx.com/company/gettingstarted/fpgavsasic.htm>

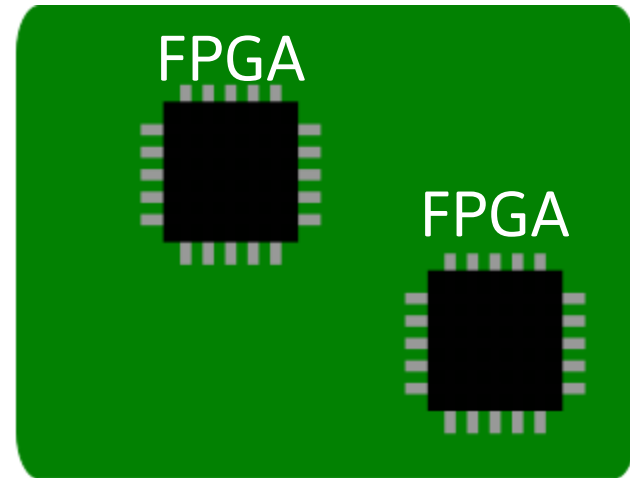
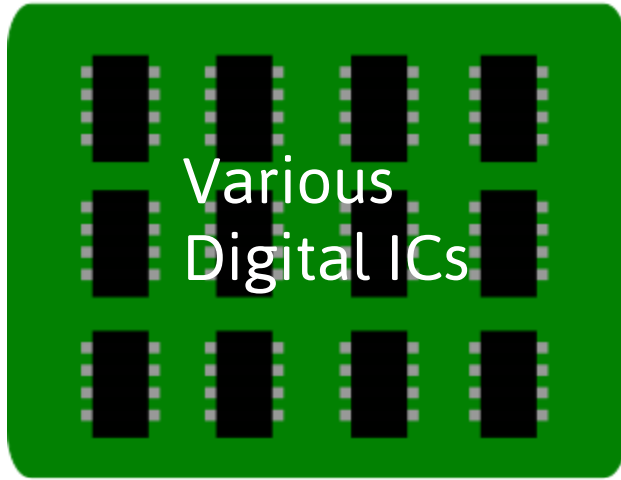


When problems arise, design iterations are required that can be very time-consuming. FPGAs have a significant advantage when design updates are required.

Applications of FPGAs

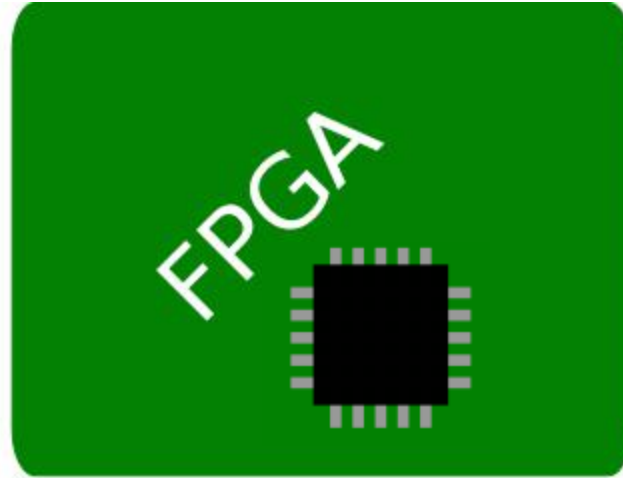
- Design Integration – many devices replaced by one
- ASIC Replacement
- Glue Logic

Applications: Integration



- Replace many ICs with one or a few FPGAs
- In general, an FPGA design will be...
 - smaller
 - cheaper to manufacture
 - consume less power
 - quicker, cheaper design
 - Streamlined & reliable design and operation
 - better assurance of part availability (don't have to stock-up on many different parts or make predictions about availability)

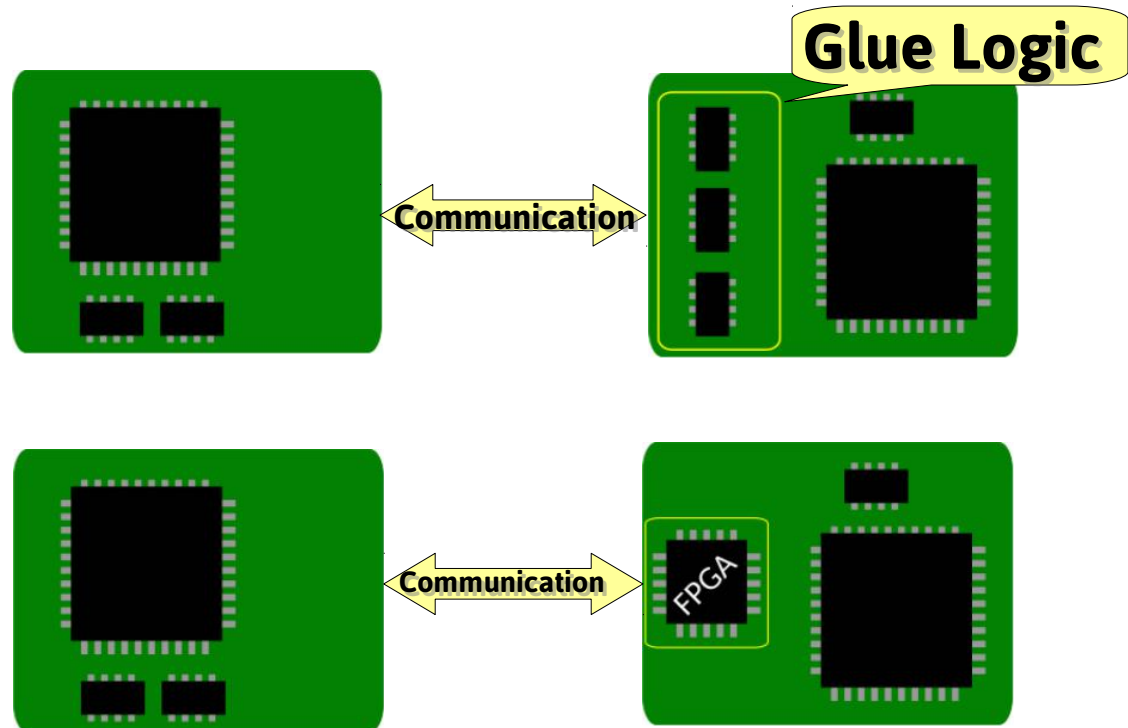
Applications: ASIC Replacement



- More likely to use FPGA for integration than ASIC
- ASIC mask production costs can be upware of 10's or 100's of thousands of \$
- ASIC design costs can be many months or years of man hours
- ASIC redesign is very expensive (must re incur mask production costs)
- FPGA allow more flexibility for specification changes
 - this facilitates better parallism of system and IC design efforts
- Can begin FPGA work and high-level board and system work in parallel and make changes according to evolving specifications.

Application: Glue Logic

- FPGAs make for good glue logic and are excellent for evolving interface specifications between design teams.
- **Glue logic** refers to misc logic required to interface multiple other primary off-the-shelf digital ICs.
- Simplest example would be inverters required to convert active low signals to active high
- More complex examples would be converting from a serial to a parallel representation
- With FPGAs, you don't need to nail down every interface detail at the start



Downsides of FPGAs

- FPGAs require some extra infrastructure versus an ASIC
 - are more expensive at high volumes versus an ASIC
 - an chip or a systems design may have to be tailored to accomidate what an FPGA can do instead of an optimal ASICs
 - designs are limited to available FPGAs so exotic system-on-IC combinations are limited.
 - Ex: can't have custom RAM+DAC+DSP (though FPGAs are getting more and more features to suit demands)
- not optimal logic power compared to custom small ASIC with custom design design and custom logic cell implementation per gate
- routing (data communication and clock) power is significant in ICs and particularly expensive in FPGAs since the logic density is less and thus designs are more spread out
- FPGAs have may have many unused components adding leakage current and power
- In spite of shortcomings, using an FPGA means capitalizing on others' high-level work (hardware designs and tools) meaning they may still be optimal given a constrained design time or time to market.

FPGA Lineage

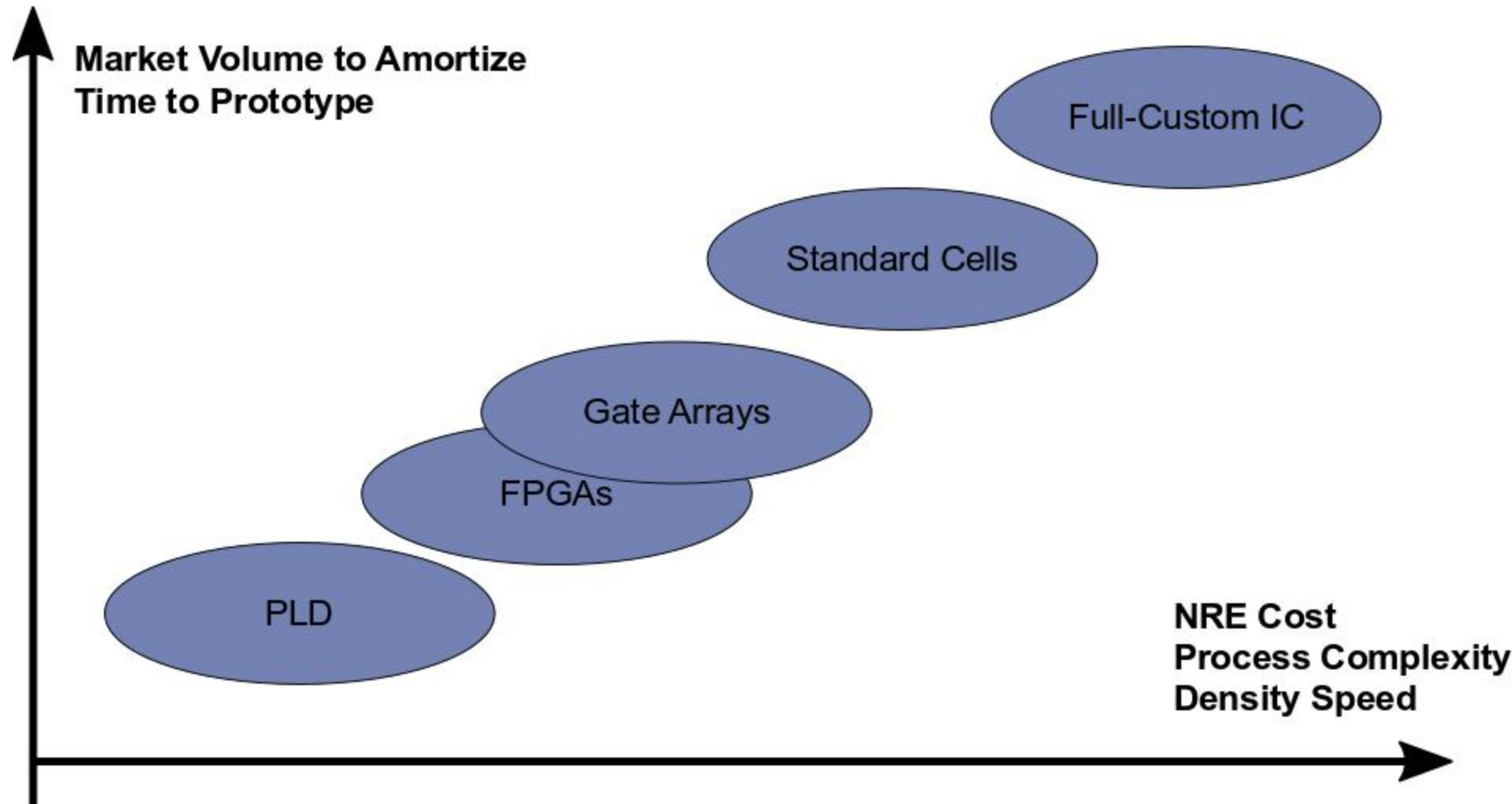
Gate Level ICs
and, or, not, etc..

PLDs (Programmable Logic Devices)
think simple and-or combinations unit

CPLDs (Complex programmable Logic Devices)
think larger&more simple and-or combination units
and interconnections

FPGAs
a variety of mentioned units and
very complex and versatile
interconnections/routing

Technology Implementation Tradeoffs



Design Entry

- Since fuse maps aren't practical for FPGA Design
- HDLs are the core of most FPGA design flows
 - Verilog, VHDL are the predominate languages
- Schematic
 - Often used at top-level to design and/or visualize ...
 - integration of modules/cores
 - visually configure some modules like PLLs (clock multipliers/dividers)
 - create or add simple glue logic
 - add/specify buffers for signals and clocks
 - specify pin assignments/types
 - Are particularly sensible when integrating different types of design sources VHDL+Verilog+Schematic+etc...
- State Diagrams - functional description, but maps well to implementation
- Higher Level Tools:
 - Try to further abstract circuit design as purely behavioural design
 - tools take care of designing the circuit
 - SystemC, Matlab, Simulink, Labview, "XYZ"-to-HDL converters, etc..
 - some provide native simulation that is very fast and convinient as compared to HDL simulation tools

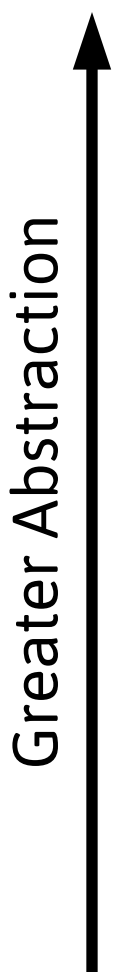
HDL advantages over Schematic

- Behavioral HDL
 - Abstraction of behavioral design process
- Top-down design and structural implementation
- Typically compact and portable
- No need for graphical display (not a big deal)
- No need for tools to create human-readable layout (visual representation)
- Repetitive/systematic changes can be easy (if you learn to use a good text editor or are handy with programming text parsing and manipulation)

HDL Code Styles

- Structural
 - Structural code is basically a text version of a schematic
 - Structural code defines instantiations and connections of primitives
 - primitives defined by simple inbuilt operators, behavioral code, switch-level models
- RTL
 - implicitly defines existence of registers (sequential logic) and the combinatorial logic functions between them
 - regarded as "synthesizable" code
- Behavioral
 - Describes function, often at highest level of abstraction
 - May or may not be synthesizable
 - Non-synthesizable code is for modeling an simulation
 - examples of non-synthesizable constructs:
 - arbitrary delay
 - Disk file IO typically during simulation

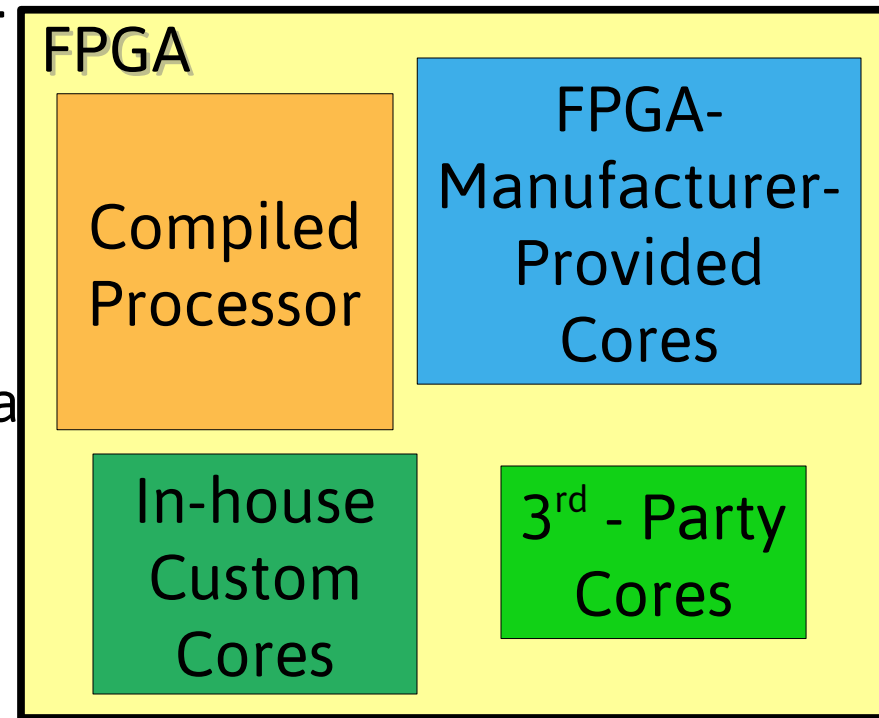
Taxonomy of Design Abstractions



LEVEL	STRUCTURAL	BEHAVIOURAL
PMS processor memory switch on a CHIP	CPU _s , MEMORIES, MICROPROCESSORS RAM, UART PARALLEL PORT	PERFORMANCE I/O RESPONSE ALGORITHMS OPERATIONS
REGISTER	REGISTERS, ALU _s COUNTERS MUXES	TRUTH TABLE STATE TABLES OPERATIONS
GATE	GATES, FLIP-FLOPS	BOOLEAN EQUATIONS
CIRCUIT	TRANSISTORS, RLC	DIFFERENTIAL EQUATIONS
SILICON	GEOMETRICAL OBJECTS	-

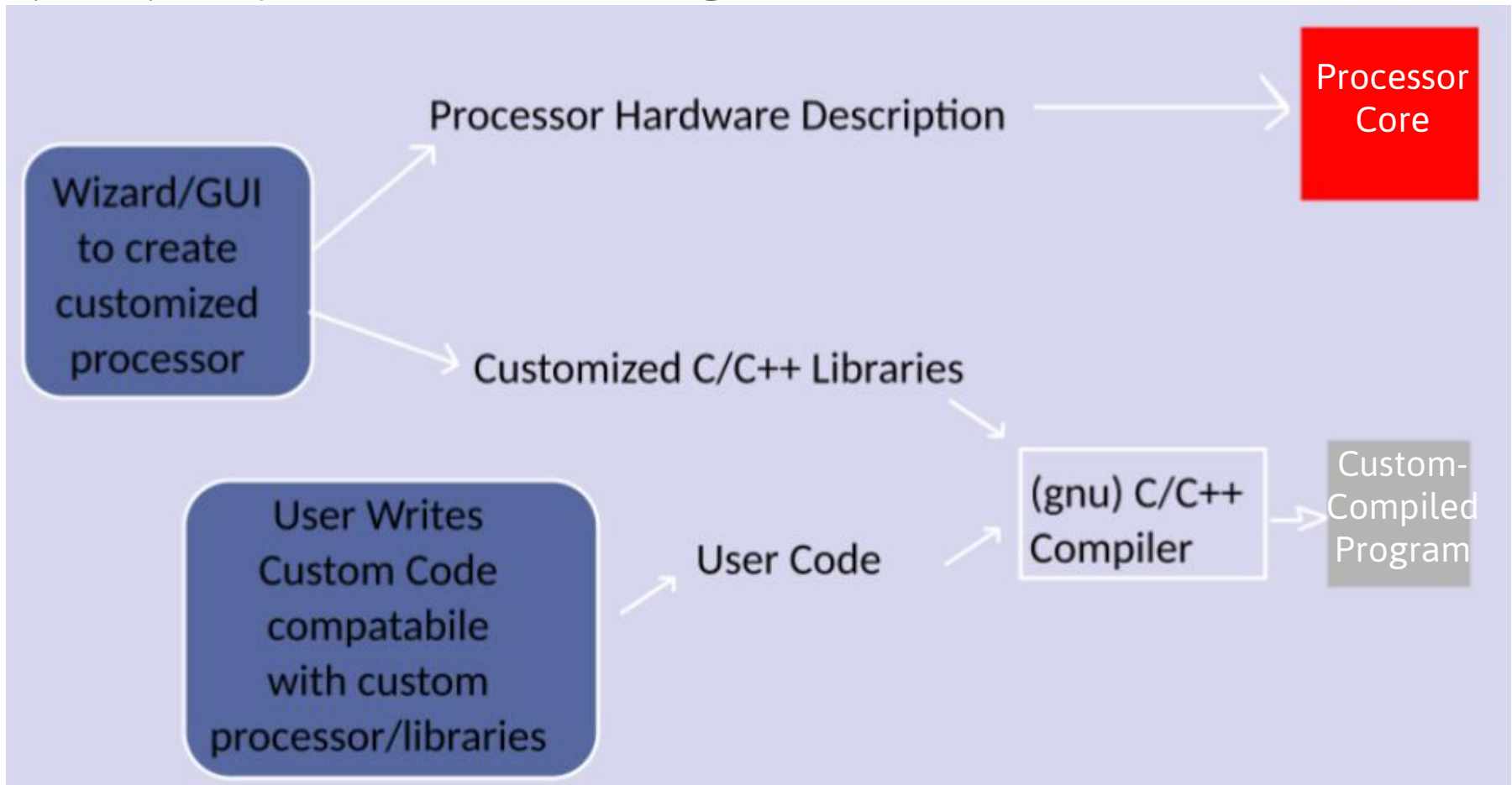
Soft Processor Development

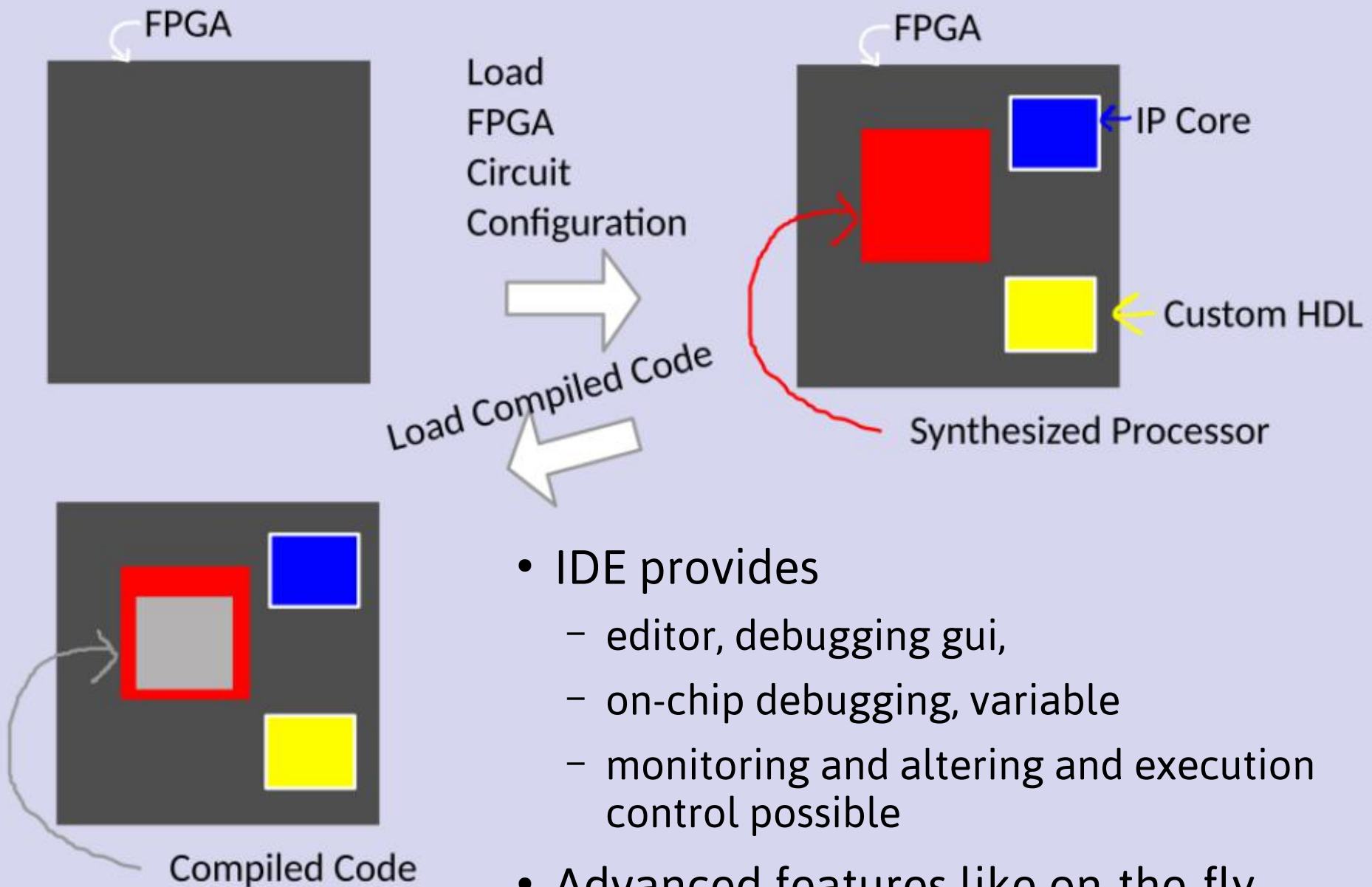
- If sequential code really is appropriate, add a processor.
- A **Soft Processor** is a compiled, customizable processor (not to be confused with fixed processors sometimes provided on FPGAs)
- examples of options and parameters:
 - cache size, #bits,
 - on-FPGA RAM,
 - external RAM controller
 - branch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



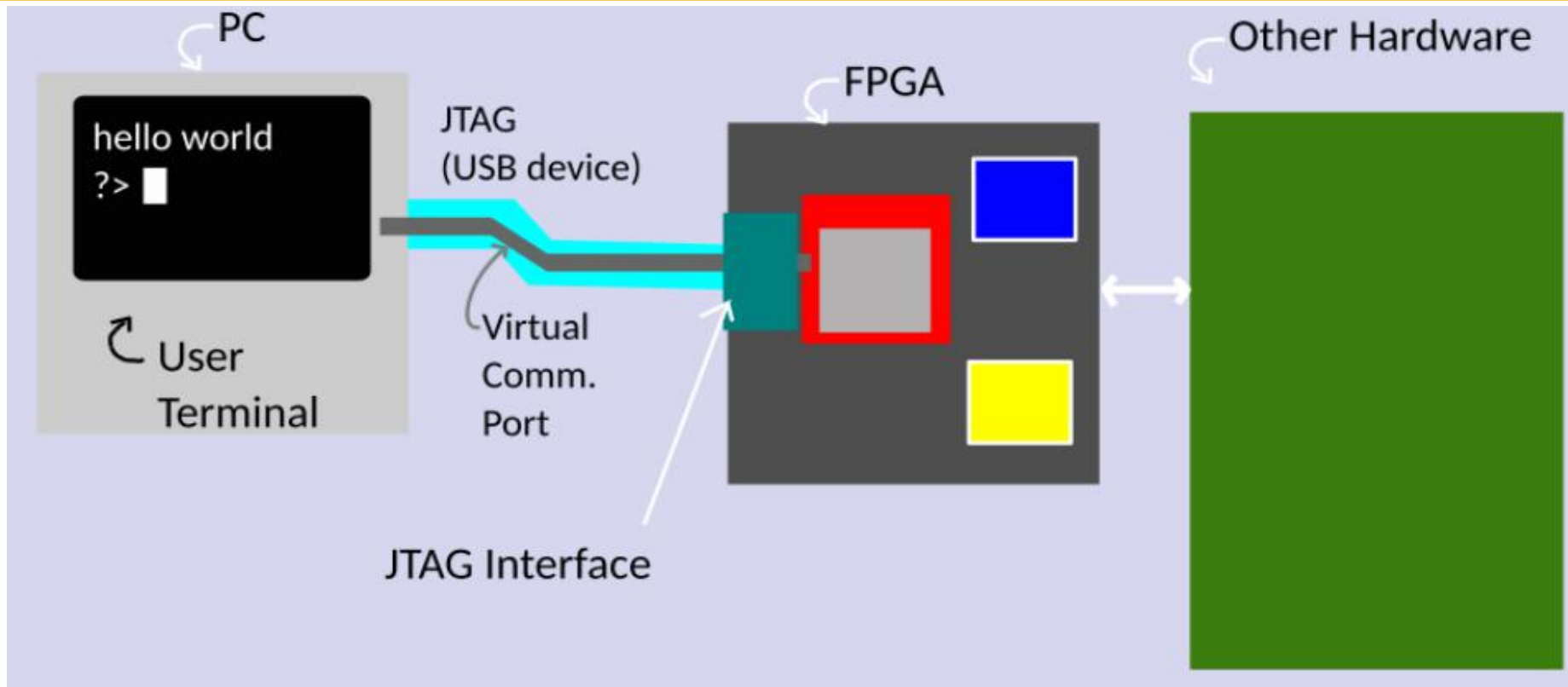


- IDE provides
 - editor, debugging gui,
 - on-chip debugging, variable
 - monitoring and altering and execution control possible
- Advanced features like on-the-fly hardware break-points depend on processor configuration

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 add-on
- Emphasizes SOC,
 - Processor and IP-Centric Design (maximally using pre-built cores)
- and
- Mixed development of C/C++ (soft processors) and other IP-based design

A Typical 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

Hardware Description Languages

- Lower-level software languages are basically used to describe algorithms using a sequential set of operations that are mapped and compiled (using a **compiler**) for sequential execution on a general purpose processor.
- **Hardware description languages** (HDL) are designed intended to simulate and model hardware. Key features of such languages are that they support for **concurrency** (multiple processing happening at once) and modeling of **time**.
- A subset of an HDL will be supported for synthesizing hardware. A **synthesis tool** maps the code to a hardware implementation. Typically the design is described with the hardware implementation in mind and somewhat mimics the structure of the hardware.
- High-level synthesis (HLS) is an automated design process that interprets an algorithmic description of a desired behavior and creates digital hardware that implements that behavior. (wikipedia)
 - High-level synthesis languages increasingly support abstract descriptions that look like a sequential operation algorithm description and are structured less and less like the hardware implementation.
 - Some support parsing code and mapping certain functions to hardware and others to compiled code to run on a processor.

FPGAs are becoming more popular

- FPGAs at a low-level exploit parallelism well beyond what general-purpose processors can do and are beginning to displace general-purpose processors in high-performance applications
- As ASIC designs become comparatively more expensive FPGAs have displaced them in the market
- Notable is
 - Intel's acquisition of Altera and
 - Amazon offering FPGA cloud computing
- Understanding FPGA technology and how to interface FPGAs with other processors is and will be a very marketable skill