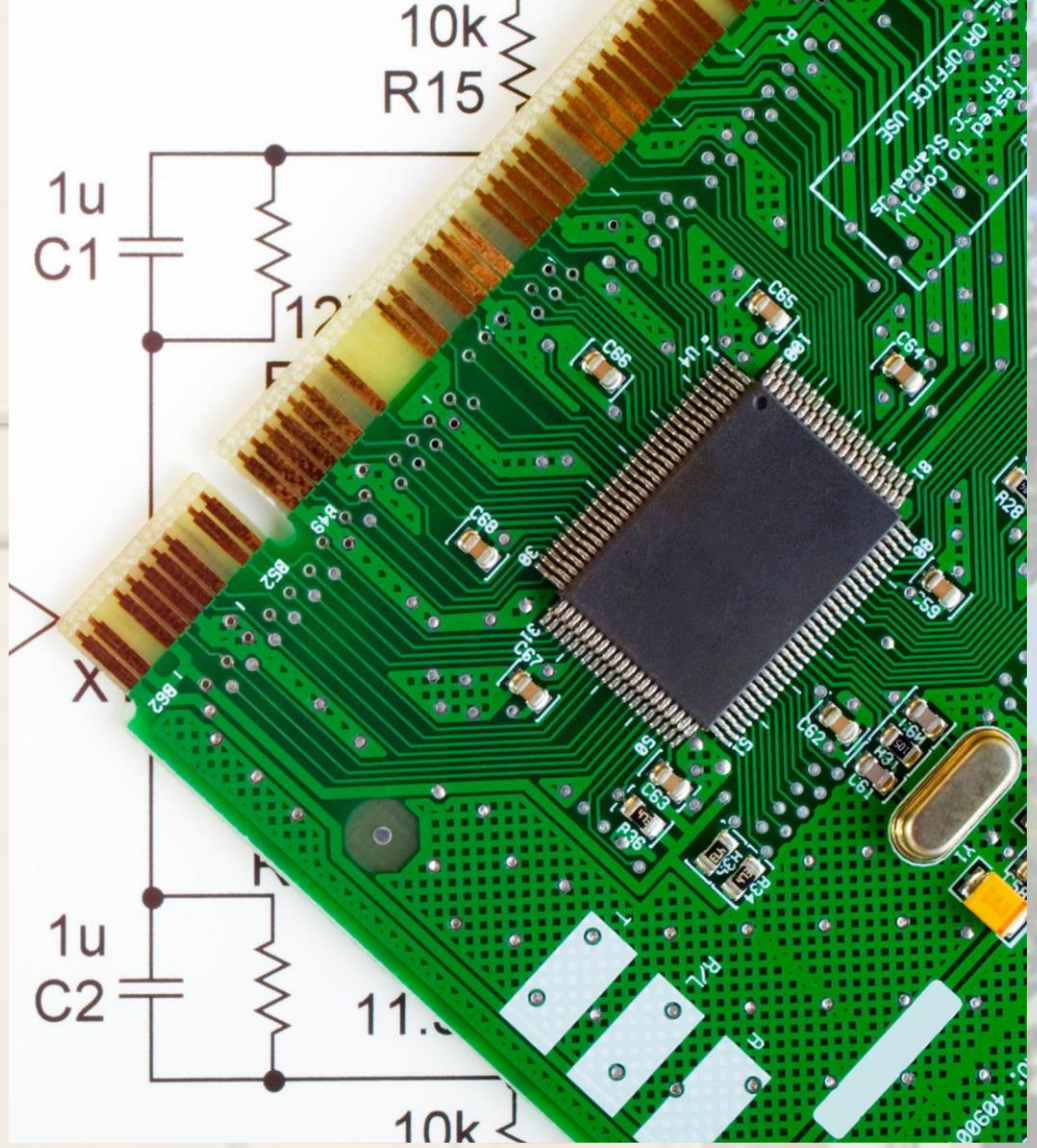
CEN 510:

Digital System Design with VHDL



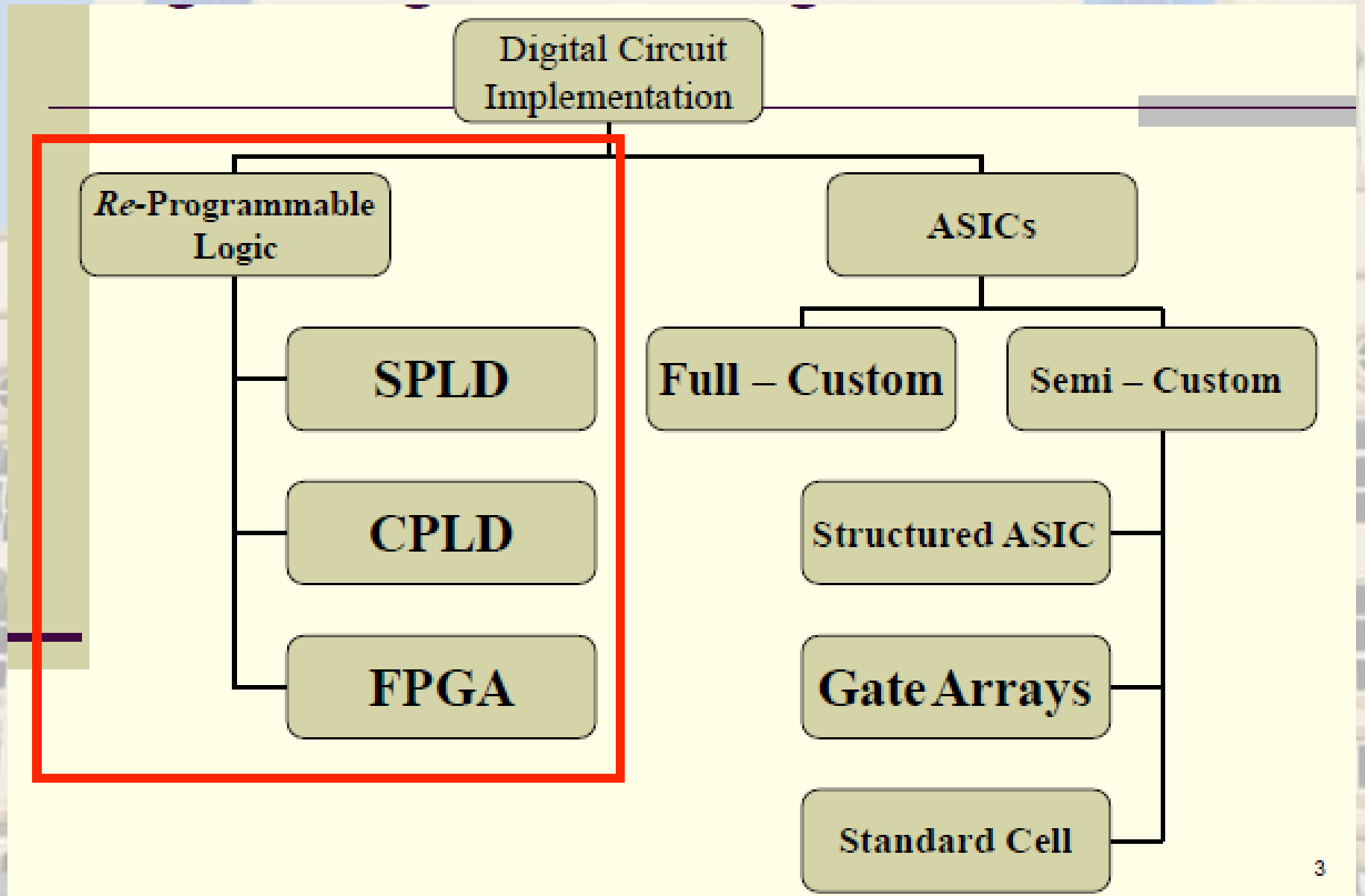
Course Lecturers: Mr. Adekunle OJEWANDE & Mr. Chibueze UBOCHI

Department of Electrical and Information Engineering
Covenant University, Nigeria

Digital System (Logic) Design

- Digital circuit or logic is basically made up of ICs.
- Logic circuits are designed based on logic (decisions).
- Logic design tools to be adopted depends largely on;
  - Device technology.
  - Boolean algebraic expression
  - Truth tables
  - Schematic capture
  - Timing diagrams
  - Logic behavioral description with languages (Verilog or VHDL)

Digital Logic Technologies



Digital System (Logic) Design

- Based on where customization (or reconfiguration) is done:
- In a fabrication facility: ASIC can be -
    - i. Full-custom ASIC
    - ii. Standard cell ASIC
    - iii. Gate array ASIC

- In the “field”: a non-ASIC is reprogrammable-
    - i. Simple programmable logic device (SPLD)
    - ii. Complex programmable logic device (CPLD)
    - iii. Field programmable gate array (FPGA)

Full-custom ASIC

- All aspects (e.g., size of a transistor) of a circuit are tailored for a particular application.
  - Circuit fully optimized
  - Design extremely complex
  - Only feasible for small components
  - Masks needed for all layers

Standard-Cell ASIC

- Circuit made of a set of pre-defined logic, known as standard cells
- E.g., basic logic gates, 1-bit adder, D FF etc
- Layout of a cell is pre-determined, but layout of the complete circuit is customized
- Masks needed for all layers

Gate-Array ASIC

- Circuit is built from an array of a single type of cell (known as base cell)
- Base cells are pre-arranged and placed in fixed positions, aligned as one-or two-dimensional array
- More sophisticated components (macro cells) can be constructed from base cells
- Masks needed only for metal layers (connection wires)

Programmable Logic Devices (PLDs)

- A PLD is an integrated circuit with internal logic gates and interconnects.
- These gates can be connected to obtain the required logic configuration.
- The term “programmable” means changing either hardware or software configuration of an internal logic and interconnects.
- The configuration of the internal logic is done by the user.
- With PLDs, the focus shifted from architecture of an actual device to hardware description methods via hardware description languages (HDL).

Why Programmable Logic Devices / Chips?

- As compared to hard-wired chips, programmable chips can be customized, according to user needs, by programming or reconfiguring.
- This *convenience*, coupled with the option of *re-programming* in case of errors, makes the programmable chips very attractive.
- Other benefits include *instant turnaround*, *low starting cost* and *low risk*.
- As compared to reprogrammable chips, ASIC (Application Specific Integrated Circuit) has a longer design cycle and were more costly.
- Still, ASIC has its own market due to the added benefit of faster performance and lower cost if produced in high volume.
- Programmable chips are good for medium to low volume products. If you need more than 10,000 chips, go for ASIC or hard copy.

Simple Field Programmable Device

- Programmable device with simple internal structure PROM (Programmable Read Only Memory)
- PAL (Programmable Array Logic)
- GAL(Generic Array Logic)
- No custom mask needed
- Replaced by CPLD/FPGA

Summary

Programmable Logic Devices (PLDs) are ICs with a large number of gates and flip flops that can be configured with basic software to perform a specific logic function or perform the logic for a complex circuit.
Major types of PLDs are:

SPLD: (Simple PLDs) are the earliest type of array logic used for fixed functions and smaller circuits with a limited number of gates. (The PAL and GAL are both SPLDs).

CPLD: (Complex PLDs) are multiple SPLDs arrays and inter-connection arrays on a single chip.

FPLD: (Field Programmable Gate Array) are a more flexible arrangement than CPLDs, with much larger capacity.

Abstraction in System Design

- **Abstraction** - the process of simplifying complex systems by focusing on high-level functionality while ignoring or hiding the detailed implementation.
- It allows designers to work at various levels of complexity, using different levels of abstraction to represent a system in a way that is easier to understand, analyze, and design.

Why Use Abstraction?

- **Simplifies Complex Systems - a**bstraction allows designers to break down large, complicated systems into manageable pieces. Instead of dealing with millions of transistors or gates, designers can work with higher-level representations like modules, registers, or entire functional blocks.
- **Improves Productivity –** by abstracting away unnecessary details, designers can focus on the higher-level structure of a design. This reduces the time spent on implementation and debugging
- **Reusability -** high-level abstractions can be reused across different designs. For example, a well-defined logic block (like an adder or a multiplier) can be reused in multiple circuits without needing to redesign it each time
- **Easier Testing and Verification -** higher-level abstractions make it easier to simulate and verify the behavior of a system before committing to lower-level designs.

Levels of Abstraction in Digital Logic Design

- **Behavioural Abstraction**

At this level, the emphasis is on what the system does (its behavior) rather than how it performs the operations. For example, in digital design, a behavior might be described as "output is the sum of two inputs," without specifying the details of how the summing is done. Designers typically use high-level hardware description languages (HDLs) like VHDL or Verilog at this level.

- **Register-Transfer Level (RTL) Abstraction**

RTL abstraction describes the flow of data between registers and the operations (like addition, subtraction, or logical operations) that occur between them. It is more detailed than behavioral abstraction but still independent of the actual implementation of gates or transistors. At this level, designers specify the data path, control signals, and registers.

Levels of Abstraction (cont'd)

- **Gate-Level Abstraction**

Here, the focus is on how the logic is implemented using basic gates (AND, OR, NOT, etc.) and other components like multiplexers, flip-flops, and decoders. This level deals with the actual gates and how they are wired together to perform the desired function

- **Transistor-Level Abstraction**

The most detailed level of abstraction, where the design is described in terms of individual transistors and their connections. This level is typically used by chip manufacturers when designing the actual silicon implementation of a system

Design Principles

Hierarchy

- Divide & conquer
- Simplification of the problem

Regularity

- Divide into identical building blocks
- Simplifies the assemblage verification

Modularity

- Robust definition of all components (entity)
- Allows easy interfacing

Locality

- Ensuring that interaction among modules remains local
- Makes designs more predictable and re-useable

Methodologies in Digital Logic Design

In digital logic design, various **methodologies** are employed to create efficient, reliable, and scalable digital systems. These methodologies represent the different approaches or strategies designers use to go from high-level specifications to the final implementation of a digital circuit. The choice of methodology often depends on the design's complexity, the tools available, the target platform (e.g., FPGA, ASIC), and the specific requirements (e.g., speed, power, area)

Design methodologies

A procedure for designing a system.

Understanding your methodology helps you ensure you didn't skip anything.

Compilers, software engineering tools, computer-aided design (CAD) tools, etc., can be used to:

- help automate methodology steps;
- keep track of the methodology itself.

Top-Down Design

- Hierarchical approach where the design starts with a high-level specification or system-level functionality and is progressively broken down into smaller sub-components or modules.

- **Advantages**:
  - Focuses on system-level behavior before getting bogged down by details.
  - Allows for modular, reusable design.
  - Facilitates team-based design by allowing different people to work on different blocks

- **Example**: Designing a microprocessor involves starting with an architectural specification and breaking it down into components like the ALU, registers, control unit, etc

Design Methodology

- Top-Down design methodology in 4 steps

1- Specifications

2- Partitioning

3- Implementation

4- Assemblage

Design Methodology

- Top-Down design methodology in 4 steps

1- Specifications

2- Partitioning

3- Implementation

4- Assemblage

Step 1: Specifications

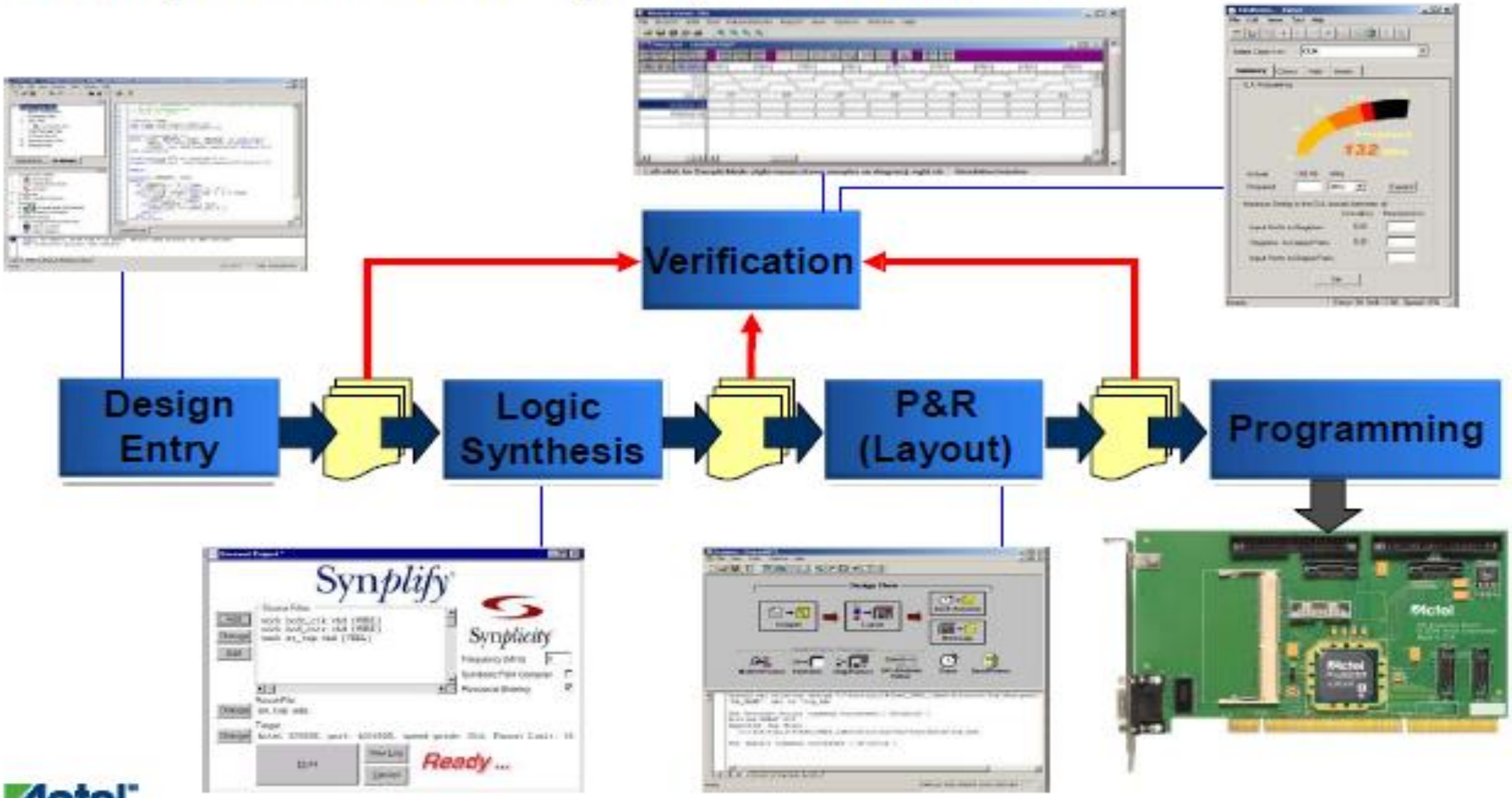
- Put down the circuit concept
  - Easy verification
  - A reference manual for communication
    - Between people
    - Between people and computers
  - How?
    - No Ordinary language
    - Accurate language
    - A language that can be simulated
- Put down the requirements
  - Timing budget
  - Power budget
  - Area budget
  - Financial budget

Step 2: Partitioning

- Divide and conquer strategy
  - Very difficult step: Relys on the know-how of the designer
  - Main idea: To split into several small parts

Step 3: Implementation

- Simplified FPGA design implementation flow



Step 4: Assemblage

- Hierarchical way
- Start from the lowest level
- Final product validation is now possible
  - Compare to original specifications
  - Simulate
  - On-board verification

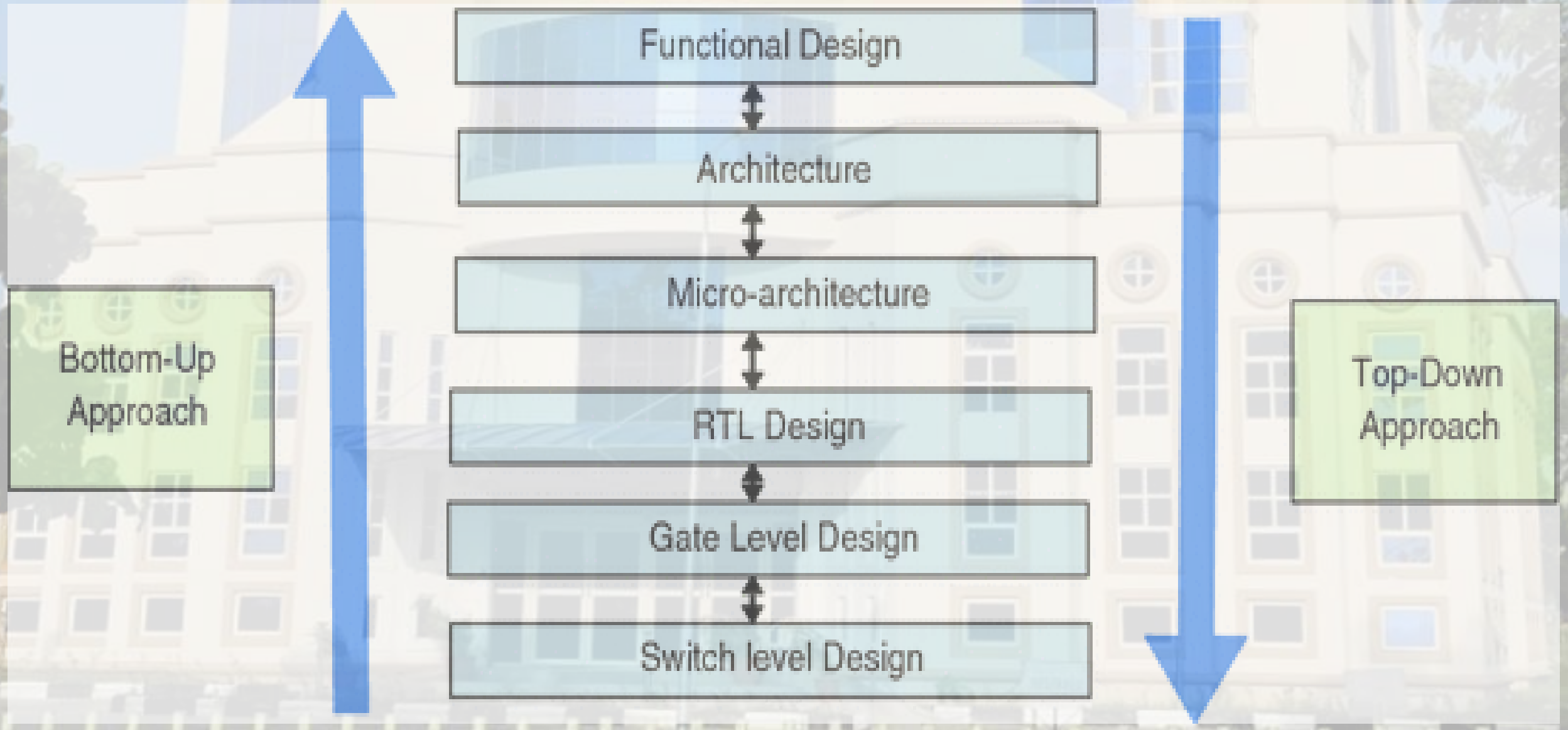
Bottom-Up Design

- System development approach that involves designing and developing individual components first, and then integrating them into a complete system. In digital system design, it may involve starting with simple, low-level components (like gates, flip-flops, or transistors) and building up to more complex systems.

- **Advantages**
  - Provides a solid understanding of the fundamental building blocks.
  - Gives the designer precise control over the circuit at every level
  - Allows for testing the functionality of the system incrementally
  - Allows for optimizing the performance of the system by leveraging existing solutions

Example: Designing a circuit using basic gates and incrementally combining them to form complex systems, such as building an ALU from individual logic gates

Digital Logic System Design Methodologies



Data-Path and Control Design

- Focuses on designing the two main parts of a digital system; the **data path** and the **control unit**.
- **Data Path**: The collection of registers, buses, ALUs, and other components that store and manipulate data
- **Control Unit**: The part of the system that directs the operation of the data path, issuing control signals that determine how data flows and how operations are executed.
- **Advantages**:
- Separates the concerns of data manipulation and control flow, making the design more manageable
- **Example**: In a microprocessor, the data path would include the ALU, registers, and buses, while the control unit would generate control signals to direct the flow of data in the data path

Finite State Machine (FSM) Design

- FSM design is a methodology used to design systems that operate in distinct states, with transitions between states triggered by inputs or clock cycles.

- **Types**:
  - **Moore Machine -** the output depends only on the current state.
  - **Mealy Machine -** the output depends on both the current state and the current input

- **Process**
  - Define the states of the system and the transitions between them.
  - Specify the input conditions that cause state transitions and any outputs produced during each state.
  - Implement the FSM using flip-flops (to store the current state), logic gates (to control transitions), and sometimes multiplexers or decoders.

FSM Design (cont'd)

- **Advantages**
  - Suitable for systems with a defined sequence of operations, such as controllers and sequential logic circuits.
  - Easy to visualize and implement, especially with tools like state diagrams.
- **Example**: Design of a traffic light controller that changes states based on a timing sequence or sensor input.

Register-Transfer Level (RTL) Design

- RTL design is a level of abstraction in which the system is described in terms of data transfers between registers and the operations performed on the data (using logic gates or arithmetic units).
- **Process**:
  - Define the flow of data between registers and the operations that take place at each clock cycle.
  - Use hardware description languages (HDLs) such as Verilog or VHDL to describe the data flow and control logic at the register-transfer level.
  - Optimize the design for performance, area, and power consumption at this level

RTL Design (Cont'd)

- **Advantages**
- Allows for clear specification of data flow and control signals.
- Suitable for both synthesis (converting RTL to gate-level) and simulation
- **Example**
- Designing a simple microprocessor where the movement of data between registers is clearly defined (e.g., moving data from one register to another or performing an arithmetic operation)

HDL-Based Design

•This methodology involves using HDLs like **Verilog** or **VHDL** to describe the behavior and structure of digital systems.

•**Process**

- Write an HDL description of the system, typically at the behavioral or RTL level.
- Simulate the design using simulation tools
- Synthesize the HDL code into a netlist that can be mapped to an actual hardware implementation (e.g., on an FPGA or ASIC)

•**Advantages**

- Allows for high-level abstraction of the design.
- Provides a direct path from design to implementation (especially for FPGAs and ASICs).
- Facilitates simulation and verification.


Thank You