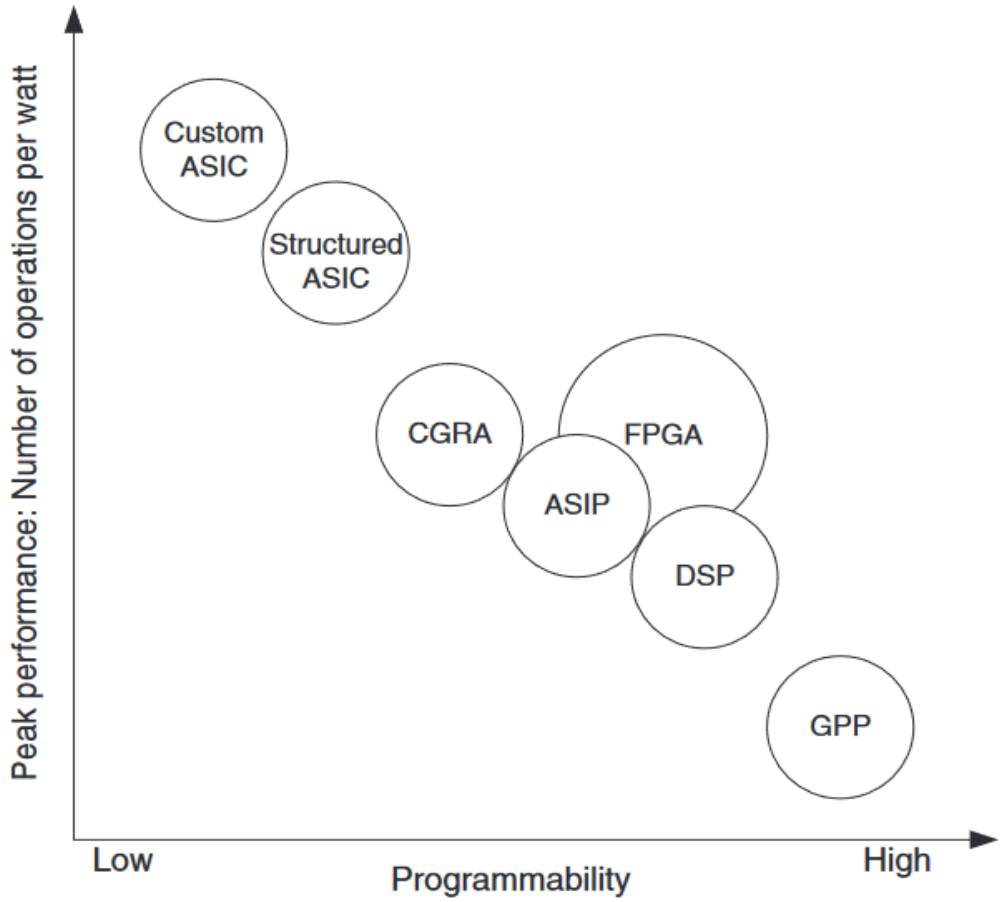
High Level Synthesis(HLS)

Various Computer Architecture



Architectures

- ASIC – Application Specific Integrated Circuit
- ASIP - Application Specific Instruction Set Processor
- CGRA – Coarse Grained Reconfigurable Array
- FPGA - Field Programmable Gate Array
- GPP – General Purpose Processor
- GPU - Graphics Processing Unit
- DSP - Digital signal processing

Intellectual Property (IP)

- An IP block is a pre-designed system component that can be used in a larger system design
- There are two major types of IP:
  - Hard IP
    - comes as a pre-designed layout.
    - Because a full layout is available, the block's size, performance, and power consumption can be accurately measured.
  - Soft IP
    - comes as a synthesizable module in a hardware description language such as Verilog or VHDL.
    - Soft IP can be more easily targeted to new technologies but it is harder to characterize and may not be as small or as fast as hard IP.

What is a High-Level Language(HLL)

- *A high-level programming language is a programming language with strong abstraction from the details of the computer*
- Software designers use high level languages (HLL) to express the algorithms in terms of language statements.
- Examples
  - Java, C/C++, Python, Java Script …

What is Hardware Description Language(HDL)

- *Digital hardware designers use hardware description languages (HDL) to describe the system they are designing.*
- *HDL's are capable of describing the parallelism, nonrecursive nature, and timing issues in the hardware more naturally, and thus differ from the pure sequential nature of a general HLL.*

- *Examples*
  - VHDL(VHSIC Hardware Description Language), Verilog, SystemVerilog, …

VHSIC - Very High Speed Integrated Circuit

What is High Level Synthesis(HLS)

High-Level Synthesis (HLS) intends to automatically convert an algorithm level specification of a hardware behavior to a register-transfer level (RTL) description that implements the behavior.

RTL

A language for describing the behavior of computers in terms of step-wise register contents

Eg: R3 = R1 + R2; // R1, R2 and R3 are registers

Benefits of HLS

- **Improved productivity for hardware designers Hardware**
- **Improved system performance for software designers**

Applications of HLS

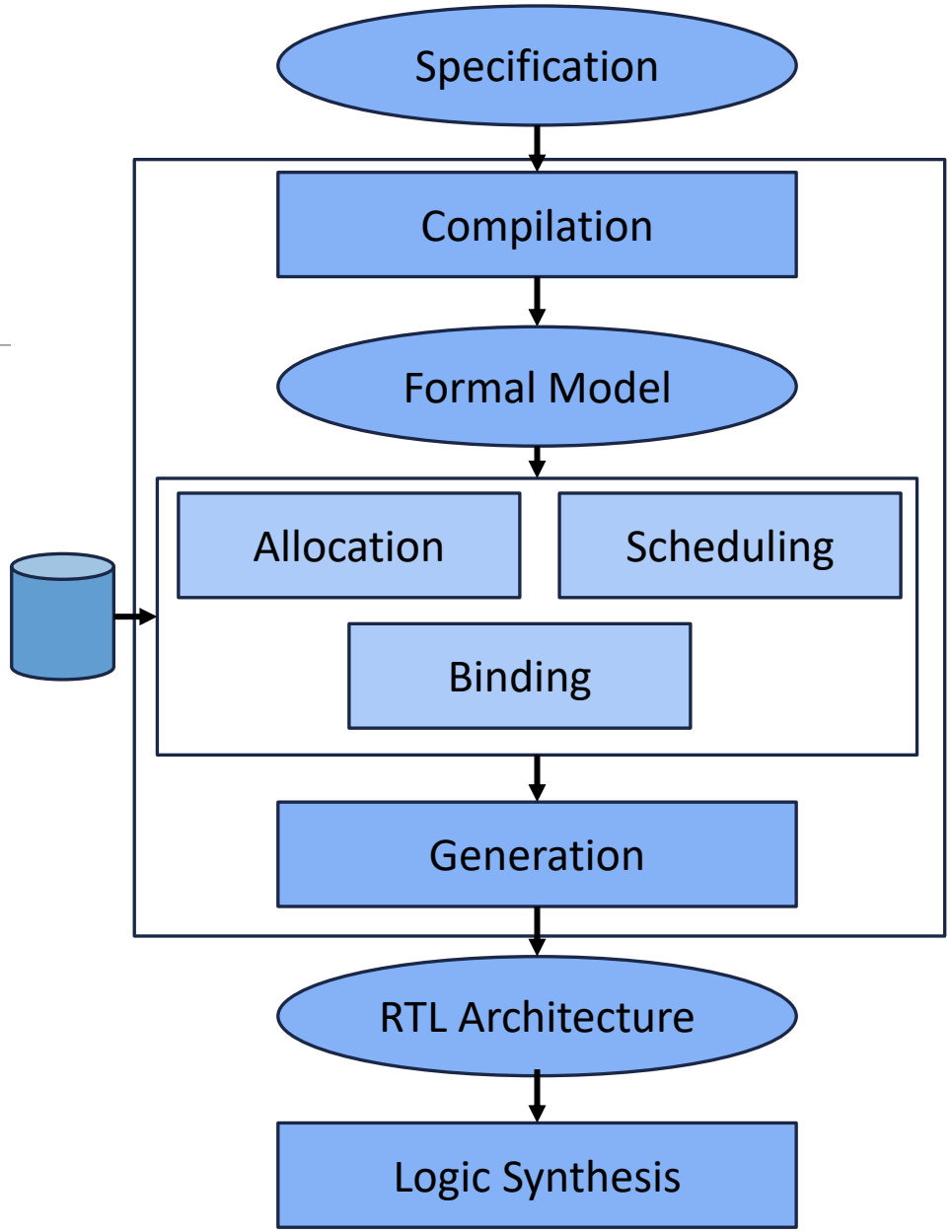
- **Signal Processing:**
  - Filters, FFTs, and image processing.
- **Machine Learning:**
  - Accelerators for AI workloads.
- **Communication Systems:**
  - Protocol implementations and modems.
- **Cryptography:**
  - Hardware accelerators for encryption.

High Level Synthesis Steps

- compiles the specification
- allocates hardware resources (functional units, storage

components, buses, and so on)

- schedules the operations to clock cycles
- binds the operations to functional units
- binds variables to storage elements
- binds transfers to buses
- generates the RTL architecture



COUSSY, P., GAJSKI, D.D., MEREDITH, M. AND TAKACH, A., 2009. AN INTRODUCTION TO HIGH-LEVEL SYNTHESIS. IEEE DESIGN & TEST OF COMPUTERS, 26(4), PP.8-17.

High Level Synthesis Steps

- Compilation and modelling
  - transforms the input description into a formal representation
  - traditionally includes several code optimizations - dead-code elimination, false data dependency elimination, and constant folding and loop transformations.
  - The formal model produced by the compilation classically exhibits the data and control dependencies between the operations.
    - Data dependencies can be easily represented with a data flow graph (DFG) in which every node represents an operation and the arcs between the nodes represent the input, output, and temporary variables.
    - The DFG model has been extended by adding control dependencies: the control and data flow graph (CDFG).
      - A CDFG is a directed graph in which the edges represent the control flow.
      - The nodes in a CDFG are commonly referred to as basic blocks and are defined as a straight-line sequence of statements that contain no branches or internal entrance or exit points

High Level Synthesis Steps

- Allocation
  - Allocation defines the type and the number of hardware resources (for instance, functional units, storage, or connectivity components) needed to satisfy the design constraints.
  - The components are selected from the RTL component library.
- Scheduling
  - All operations required in the specification model must be scheduled into cycles.
- Binding
  - Every operation in the specification model must be bound to one of the functional units capable of executing the operation
  - Storage and functional unit binding also depend on connectivity binding, which requires that each transfer from component to component be bound to a connection unit such as a bus or a multiplexer

High Level Synthesis Steps

- RTL Architecture
  - The RTL architecture is implemented by a set of register-transfer components.
  - It usually includes a controller and a data path.
  - A data path consists of a set of storage elements (such as registers, register files, and memories), a set of functional units (such as ALUs, multipliers, shifters, and other custom functions), and interconnect elements (such as tristate drivers, multiplexers, and buses)
  - The controller is a finite state machine that orchestrates the flow of data in the data path by setting the values of control signals (also called control word) such as the select inputs of functional units, registers, and multiplexers.

Common HLS Tools

- Xilinx Vivado HLS
- Intel HLS Compiler
- Cadence Stratus HLS
- Mentor Graphics Catapult HLS

Vivado

- Vivado Design Suite is a software suite for synthesis and analysis of hardware description language (HDL) - Wikipedia
  - Vivado HLx -> High -Level Edition
    - Vivado HLS
    - Vivado
    - Vivado SDK/Vitis

Vivado HLS

The Xilinx Vivado HLS tool synthesizes a C function into an IP block that you can integrate into a hardware system

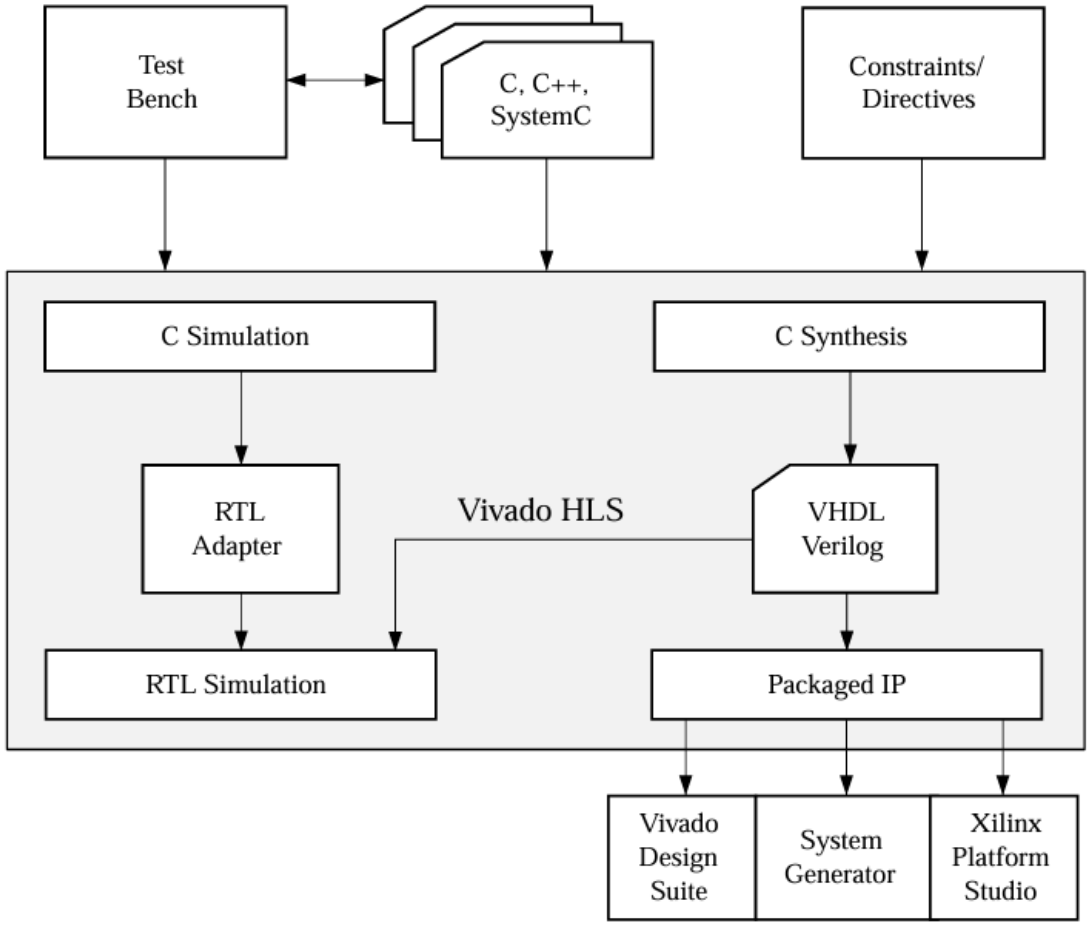
HLS Inputs & Outputs

- Inputs
  - C function written in C, C++, or SystemC
    - This is the primary input to Vivado HLS. The function can contain a hierarchy of sub-functions.
  - Constraints
    - Constraints are required and include the clock period, clock uncertainty, and FPGA target. The clock uncertainty defaults to 12.5% of the clock period if not specified.
  - Directives
    - Directives are optional and direct the synthesis process to implement a specific behavior or optimization.
  - C test bench and any associated files
    - Vivado HLS uses the C test bench to simulate the C function prior to synthesis and to verify the RTL output using C/RTL Cosimulation.

HLS Inputs & Outputs

- Outputs
  - RTL implementation files in hardware description language (HDL) formats
    - VHDL (IEEE 1076-2000)
    - Verilog (IEEE 1364-2001)
  - Report files

Vivado HLS Design Flow



Language support

- Vivado HLS supports the following standards for C compilation/simulation:
  - ANSI-C (GCC 4.6)
  - C++ (G++ 4.6)
  - SystemC (IEEE 1666-2006, version 2.2)

Note

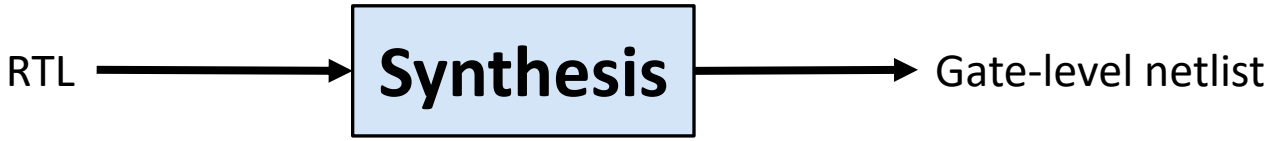
- In the Vivado HLS design flow, you can specify any sub-function below main() as the top-level function for synthesis. You cannot synthesize the top-level function main().
- The C test bench includes the function main() and any sub-functions that are not in the hierarchy under the top-level function for synthesis.
- Following are additional rules:
  - Only one function is allowed as the top-level function for synthesis.
  - Any sub-functions in the hierarchy under the top-level function for synthesis are also synthesized.
  - If you want to synthesize functions that are not in the hierarchy under the top-level function for synthesis, you must merge the functions into a single top-level function for synthesis.

Overall Flow

1. Create a project with an initial solution.
2. Verify the C simulation executes without error.
3. Run synthesis to obtain a set of results.
4. Analyze the results.

Synthesis

Synthesis is the process of transforming an RTL-specified design into a gate-level representation



Vivado IP Design and System-Level Design Integration

The Vivado Design Suite provides an environment to configure, implement, verify, and integrate IP as a standalone module or within the context of the system-level design