High-level synthesis

Besjana Jaçaj

January 11, 2024

History

- Software domain
    - Machine code
    - 1950s: Assembly language
    - HLLs

- Hardware domain
    - 1960s: ICs designed, optimized, and laid out by hand
    - early 1970s: Gate-level and cycle-based simulation
    - 1980s: schematic circuit capture, formal verification and static timing analysis
    - HDLs - Verilog(1986) and VHDL (1987)
    - 1990s: First generation of HLS tools
    - ...
    - [1] [2]

Informal definition

High-Level Synthesis is an **automated design process** that takes an *abstract* **behavioral specification** of a digital system and generates a **register-transfer level** structure that realizes the given behavior.[3]

Informal definition pt2 [3]

HLS

- **as a design process**
    - Macro Architecture
    - Design Intent
    - Constraints
    - **as a tool (automation provided by)**
        - FSM Generation
        - Schedule of Operations
        - Clock, Pipelining Registers (Micro Architecture)
        - Sharing Resources
        - Timing
        - Verification

Benefits of HSL

- **Productivity Gains**
  - are easier to write since they contain few implementation details
    - create high-quality(optimized) and error-free RTL quickly
  - are smaller and less complex than equivalent RTL descriptions
  - are easier to understand and debug since they are more closely related to the algorithm being developed
  - are faster to simulate (typically) than an equivalent RTL description, promoting early system verification
- **Architecture / Goal Flexibility**
  - enable item re-use
  - reduce verification time

Graphical definition

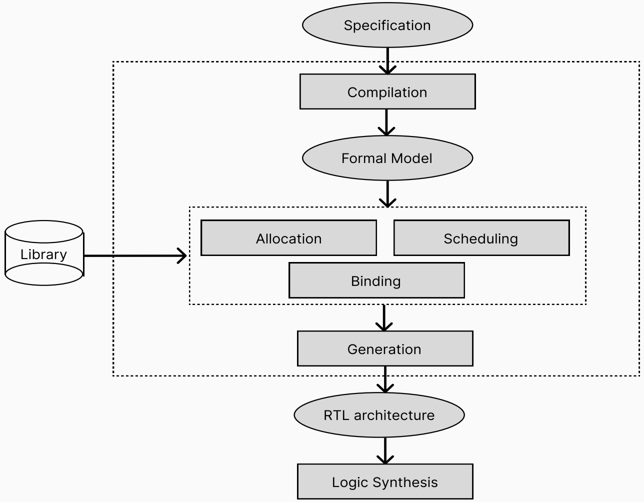


Figure 1: High-level synthesis design steps [4]

Input languages

Input specification written in HLLs such as :

- ANSI C, C++
- System C
- ...

for HTL **tools** such as :

- Catapult C
- Cynthesizer
- CyberWorkbench
- PICO
- C-to-Sicilon
- ...

Allocation step

Resource allocation is the process of deciding how many and which kind of resources can be used to a given implementation. [2]

Said resources:

- Functional units
- Storage components
- Connectivity components (such as buses or point-to-point connections)

The components are selected from the RTL component library.

Scheduling step

All operations required in the specification model must be scheduled into cycles.

Scheduling algorithms can be grouped into:

- unconstrained design
- constrained design

What are the constraints ? A design can be **resource-constrained, time-constrained, or both**.[1]

Example: a = b *op* c

Depending on the functional component :

- operation can be scheduled within one clock cycle (or more)
- operations can be **chained**
- operations can be scheduled to **execute in parallel**

Binding step

STORAGE binding: Each **variable** that carries values across cycles must be bound to a storage unit.

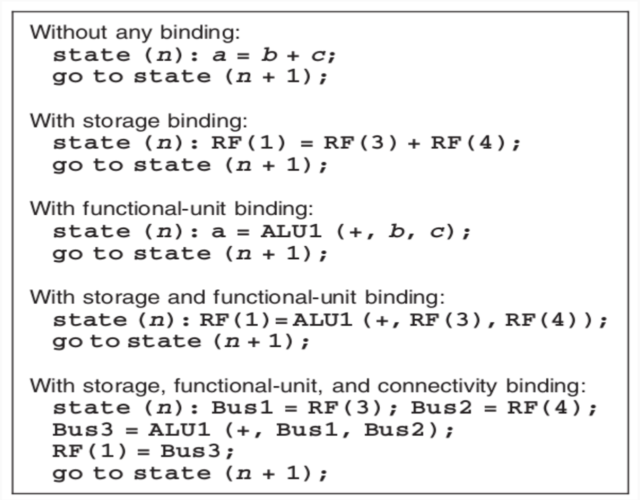
- several variables with mutually exclusive lifetimes can be bound to the same storage units

FUNCTIONAL binding: Each **operation** in the specification must be bound to one of functional units (capable of executing said instruction)

- more than one unit with same capability, the binding algorithm must optimized the selection

CONNECTIVITY binding: Each **transfer** from components to component is bound by a connection unit

Binding step



Without any binding:

```
  state (n): a = b + c;
  go to state (n + 1);
```

With storage binding:

```
  state (n): RF(1) = RF(3) + RF(4);
  go to state (n + 1);
```

With functional-unit binding:

```
  state (n): a = ALU1 (+, b, c);
  go to state (n + 1);
```

With storage and functional-unit binding:

```
  state (n): RF(1)=ALU1 (+, RF(3), RF(4));
  go to state (n + 1);
```

With storage, functional-unit, and connectivity binding:

```
  state (n): Bus1 = RF(3); Bus2 = RF(4);
  Bus3 = ALU1 (+, Bus1, Bus2);
  RF(1) = Bus3;
  go to state (n + 1);
```

Figure 2: RTL description written with different binding details. [4]

Some commonly used abbreviations

- HDL - Hardware description language
- HSL - High-level synthesis
- HLL - High-level languages
- FSM - Finiti State Machine
- IC - Integrated chip
- RTL - Register-transfer level

References i

References

[1] D. Knapp, Behavioral Synthesis: Digital System Design Using the Synopsys Behavioral Compiler. Prentice Hall, 1996.

[2] J. Elliot, Understanding Behavioral Synthesis: A Practical Guide to High-Level Design. Kluwer Academic Publishers, 1999.

[3] X. Inc., **Vitis High-level Synthesis User Guide,**, [Online]. Available: https://docs.xilinx.com/r/en-US/ug1399-vitis-hls/Introduction.

[4] P. Coussy, D. Gajski, M. Meredith, and A. Takach, **An introduction to high-level synthesis,** Design Test of Computers, IEEE, vol. 26, pp. 8–17, Sep. 2009. DOI: 10.1109/MDT.2009.69.