TU WIEN Informatics

Advanced Computer Architecture

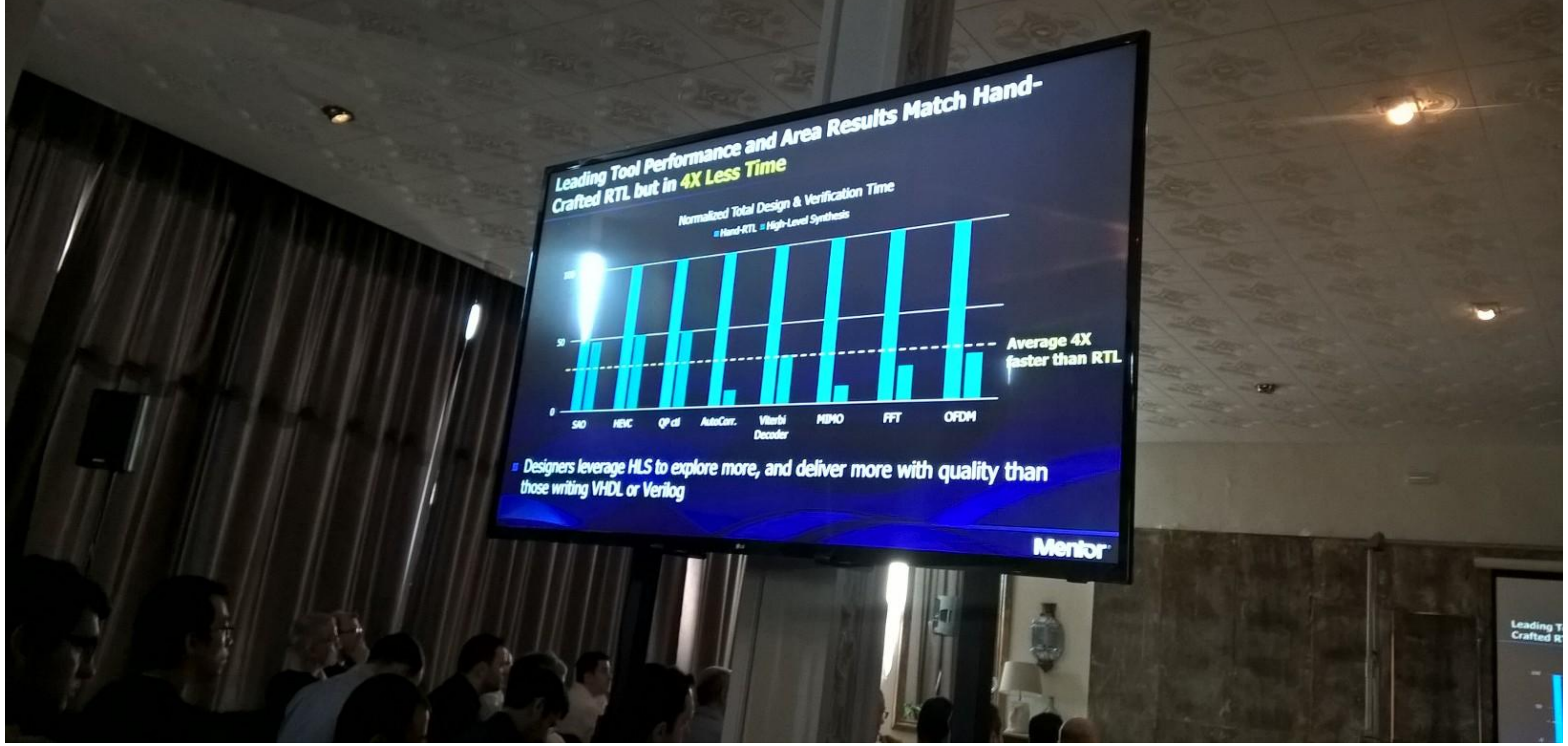
D1 – Introduction to High Level Synthesis (HLS)

Daniel Mueller-Gritschneder

Motivation for HLS

Source: WALDEN C. RHINES
President and Chief Executive Officer , Mentor, a Siemens Business
24th IEEE International Symposium on On-Line Testing and Robust System Design 2018

High-Level Synthesis: 4x faster than RTL
RTL Bottleneck: Verification



Motivation for HLS

Source: WALDEN C. RHINES
President and Chief Executive Officer , Mentor, a Siemens Business
24th IEEE International Symposium on On-Line Testing and Robust System Design 2018

High-Level Synthesis is the future of IC design

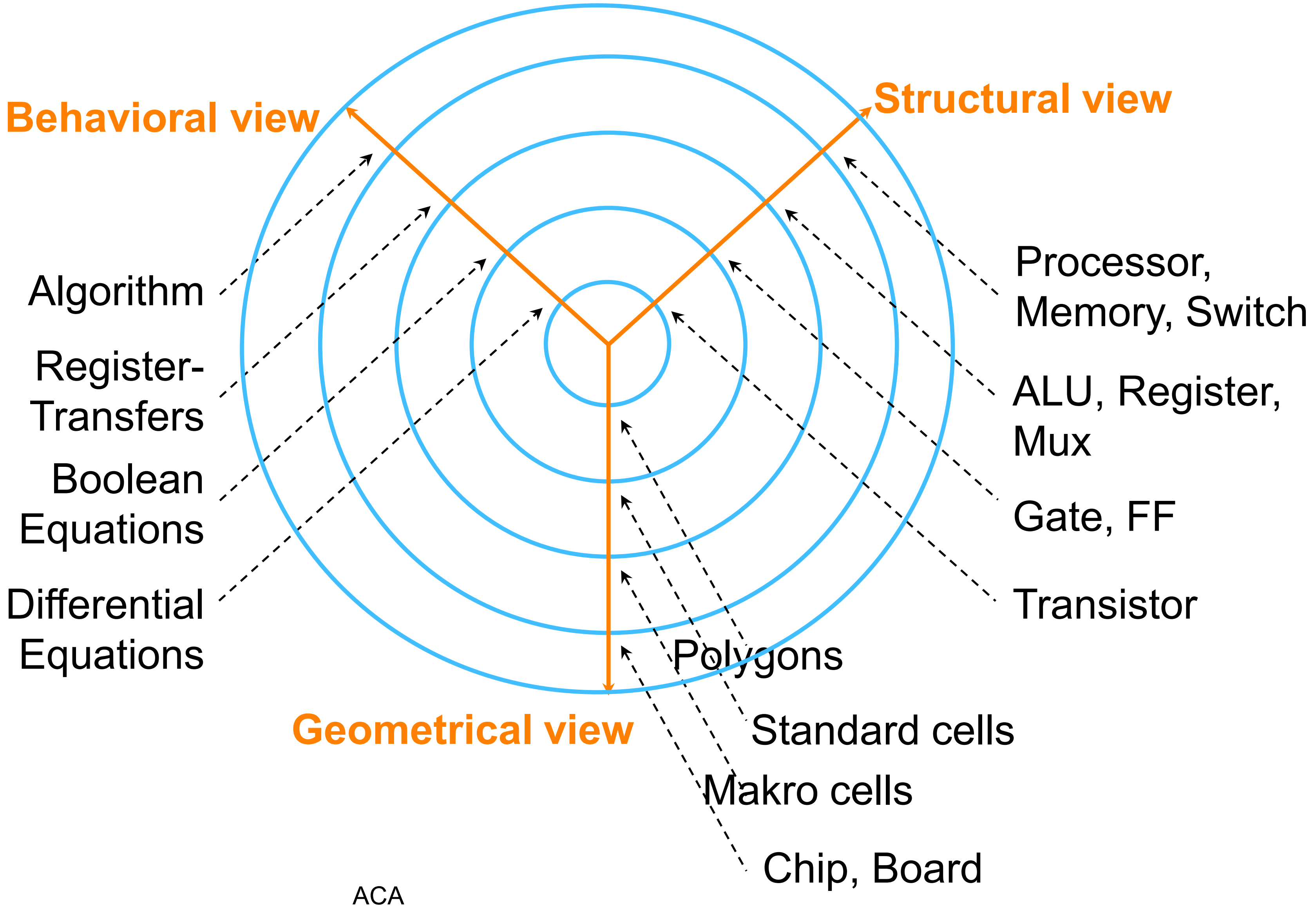
D1-1 HW Design Flow in a Nutshell

- Literature:
- „*Specification and Design of Embedded Systems*“ Daniel D. Gajski, Prentice Hall 1994
- „*Digitale Hardware/Software Systeme*“, Jürgen Teich, Springer 1997
- „*Embedded System Design*“, Daniel D. Gajski et.al., Springer 2009

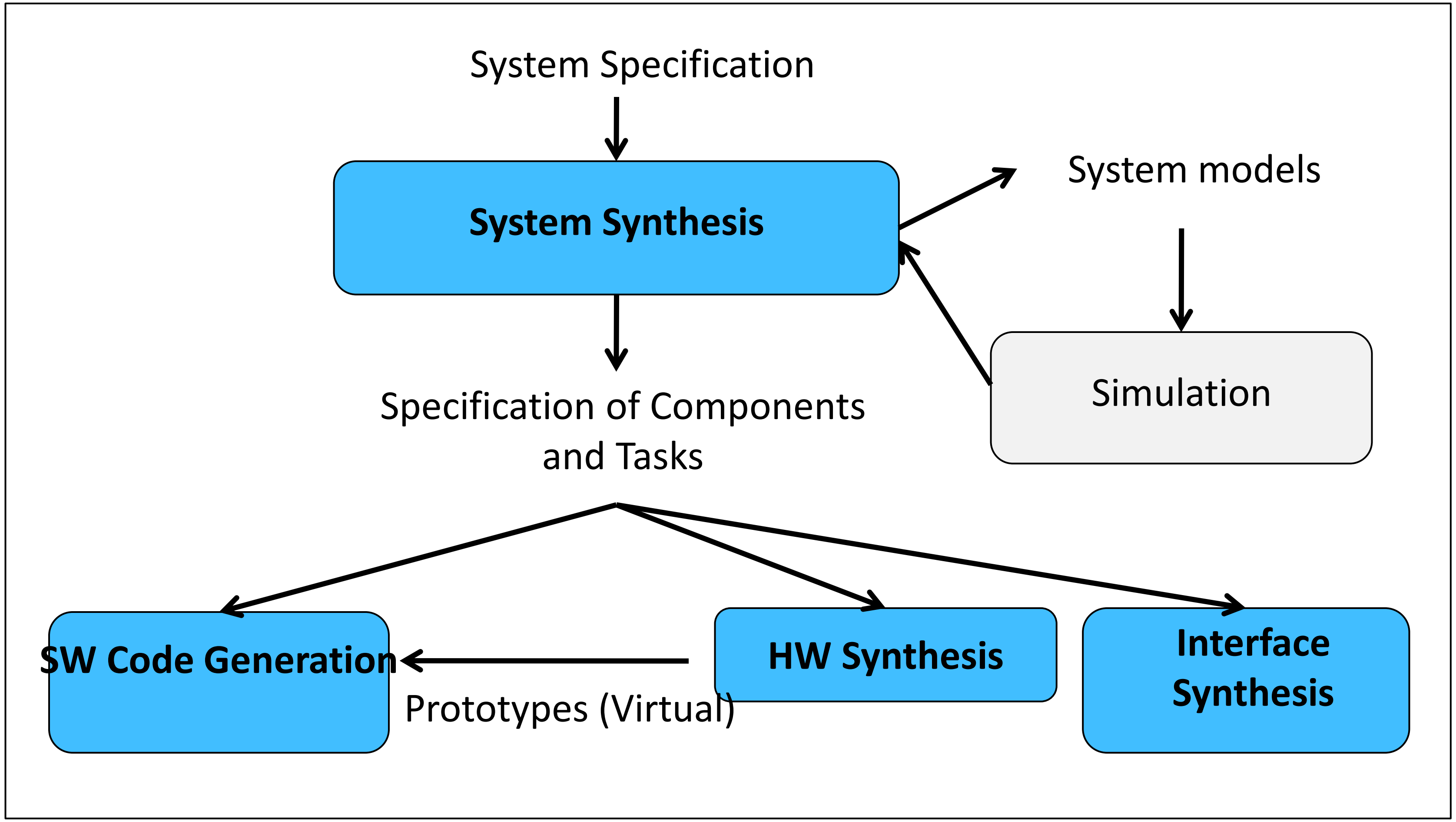
Abstraction Levels & Design Views

		Design View		
		Behavior	**Structure**	**Geometry**
Abstraction Level	**System**	System Specification	Connected Components	Chip, Board
	Architecture	Algorithms	CPU, Bus, HW-accelerator	Floor plan
	Register Transfer	Register Transfers / FSMs	Module netlist (ALU, Mux, Register)	Makro-cells (IP-blocks)
	Logic	Boolean Equations	Gate netlist (Gates, FlipFlops)	Standard cells, library cells
	Circuit	Differential Equations	Transistor netlist	Mask data

Abstraction Levels & Design Views



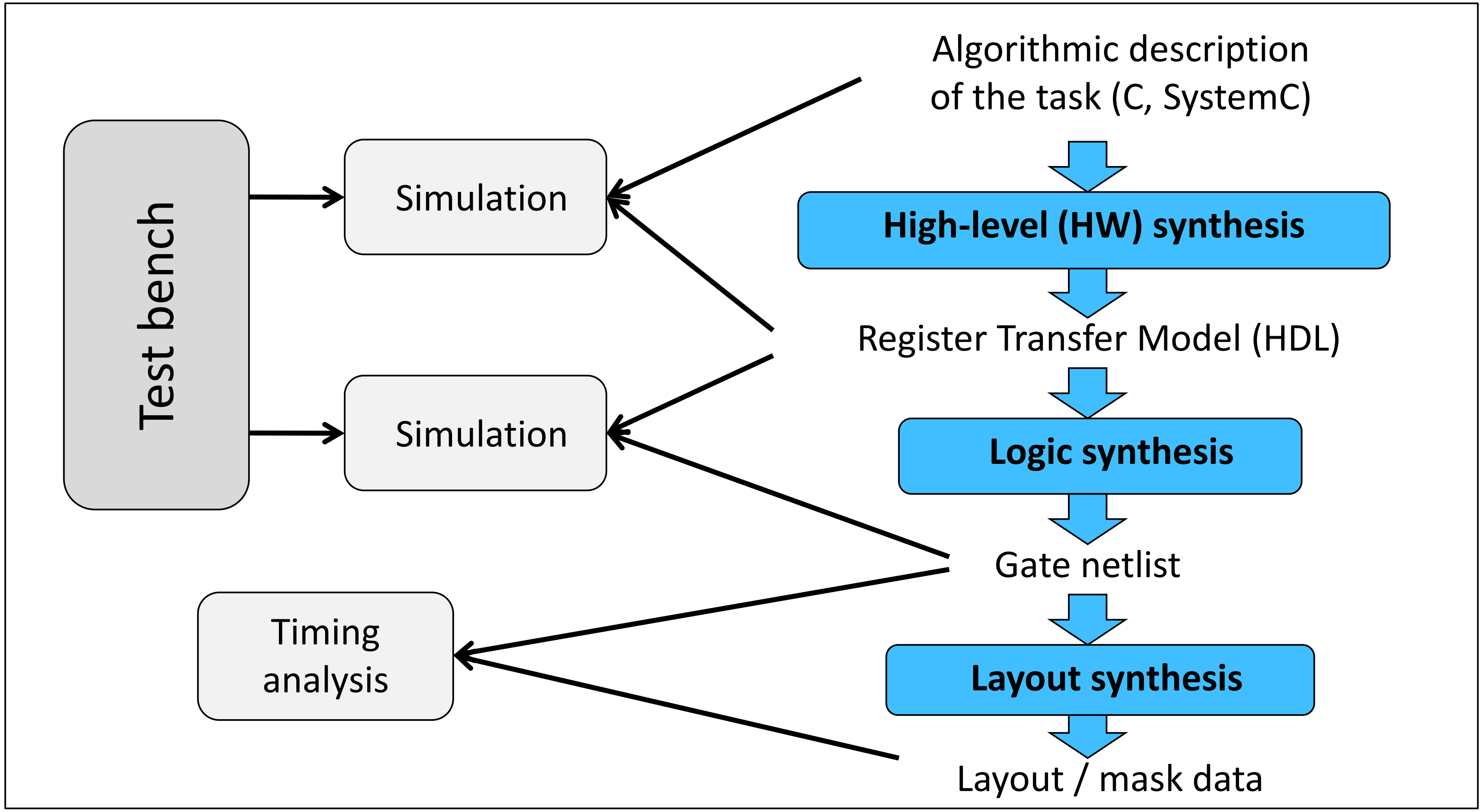
Electronic System Level (ESL) Design Flow



System Synthesis

- **Inputs:**
  - Specification of the System: Description of the functionality and design constraints (Written text, specification languages)
- **Typical synthesis steps:**
  - Description of functionality as set of communicating g tasks
  - Description of behavior of tasks on algorithmic level
  - Description of task communication
  - Allocation of system components such as processors, buses, memory, … Buses, memory,…
  - Binding of tasks and inter-task communication to system components (HW/SW Partitioning)
- **Output**
  - Specification of components, tasks and Inter-task communication that guarantees to meet system specification

ASIC HW Synthesis Flow



HLS Synthesis Step

- **Input**
  - Algorithmic description of a task, e.g. in C, C++, SystemC
  - Design constraints (Maximal latency, available resources, …)

- **Synthesis steps:**
  - Static code analysis and code optimization
  - Data path synthesis (Scheduling, allocation, binding)
  - Control unit synthesis (FSM implementation)

- **Output:**
  - Description of hardware module on RT level

Logic Synthesis Step

- **Input:**
  - Description of HW module on RT level
  - Design constraints (minimal clock frequency, maximal area,…)
  - Gate library
- **Synthesis steps:**
  - Logic optimization
  - Technology mapping
- **Output:**
  - Gate netlist

Physical Synthesis Step (Layout and Routing Step)

- **Input**
  - Gate library
  - Design constraints
  - Layout library (P-cells)
- **Synthesis steps:**
  - Placement of modules
  - Routing of signal nets
- **Output**
  - Layout, mask data

Software Compilation

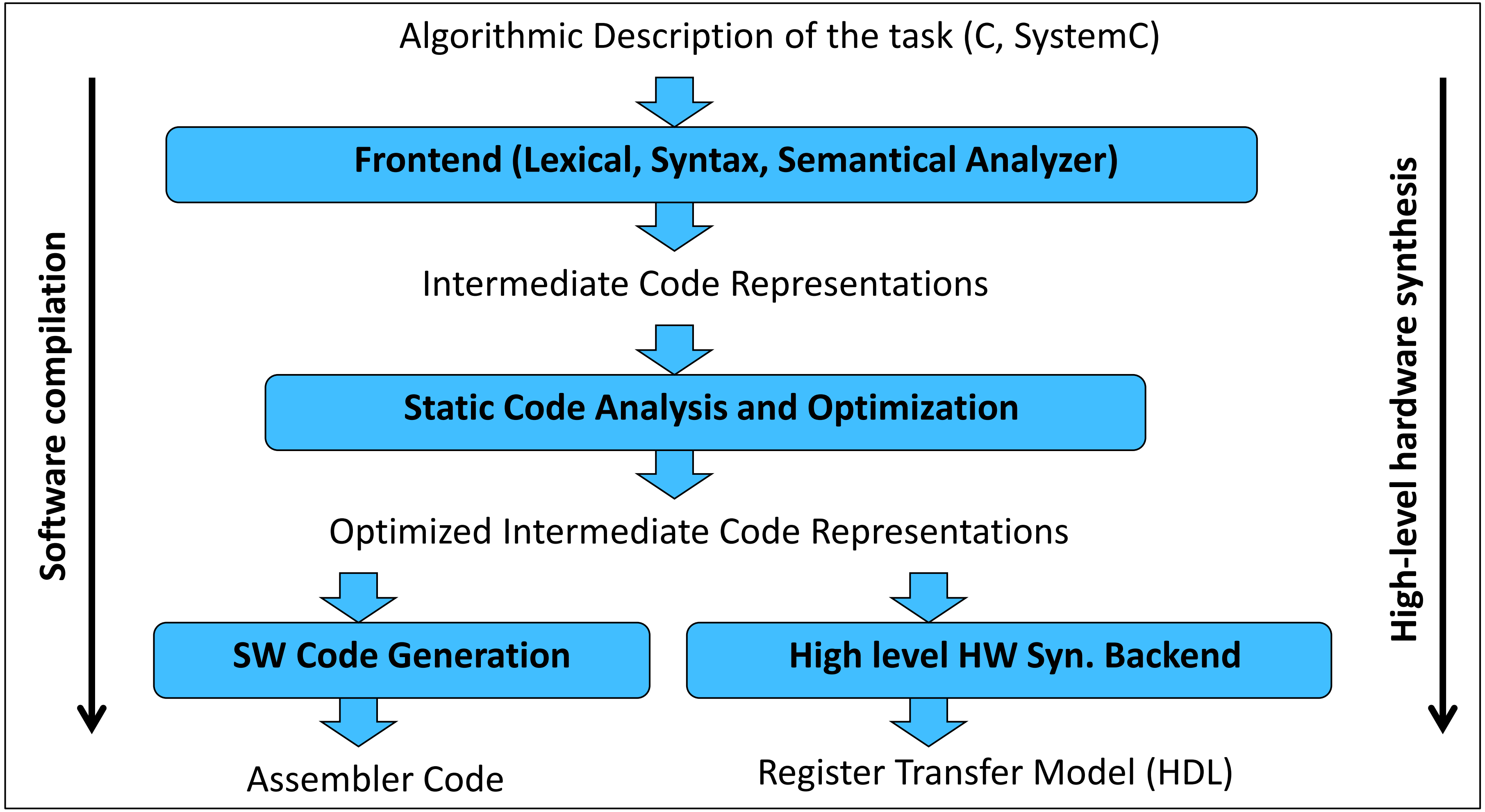
- Inputs
  - Algorithmic description of task
- Synthesis steps
  - Static Code Analysis and Optimization
  - Code Generation (Instruction Selection, Register Allocation and Assignment)
  - Assembler/linker/loader
- Outputs:
  - Assembly code/machine code for target processor

Interface Synthesis Step

- Input
  - Description of Inter-task communication
  - Design constraints (protocols, data rates, …)

- Outputs
  - Drivers, bus interfaces, …

High level HW Synthesis (HLS) vs. SW Compilation Flow



D1-2 The HLS Synthesis Task

- Literature:
- „*Specification and Design of Embedded Systems*“ Daniel D. Gajski, Prentice Hall 1994
- „*Digitale Hardware/Software Systeme“*, Jürgen Teich, Springer 1997
- „*Embedded System Design*“, Daniel D. Gajski et.al., Springer 2009

Basic Task

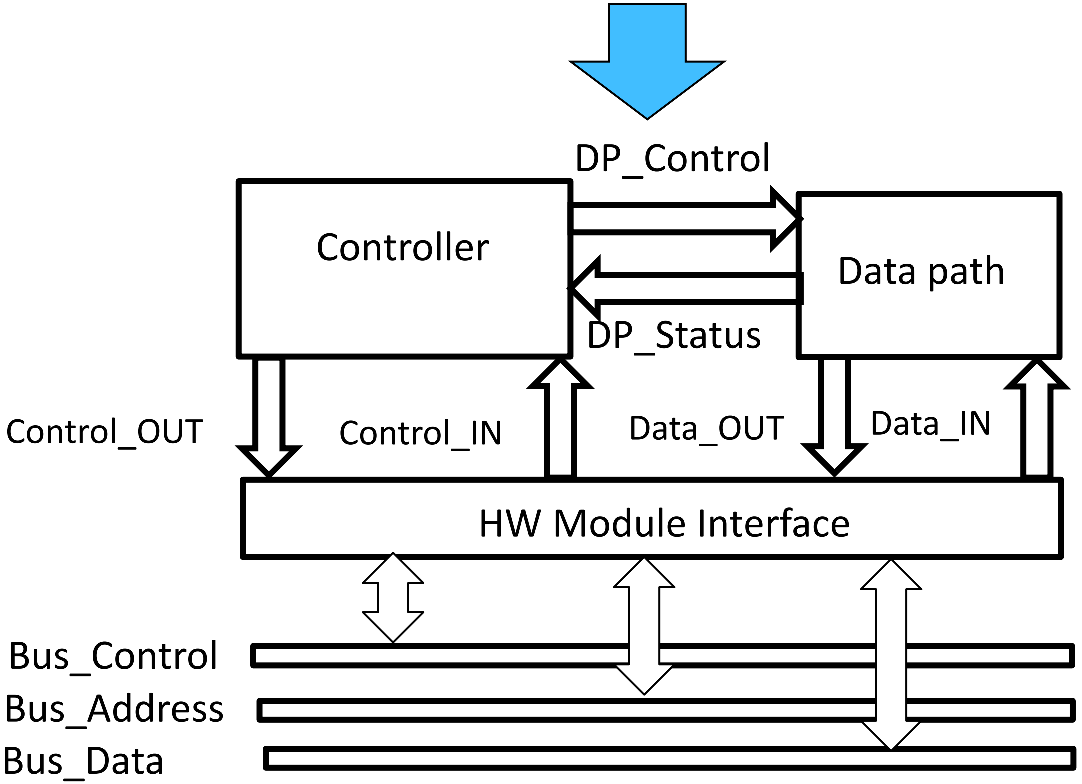
Algorithmic description of the task (C, SystemC)

High-level HW Synthesis

RT model of Hardware module in VHDL or Verilog

This is called usually an HW accelerator or IP block

```
int function1(int x, int y, int z)
{
int a;
a=x*(y*y+z);
return a;
}
```



Other Names for HLS

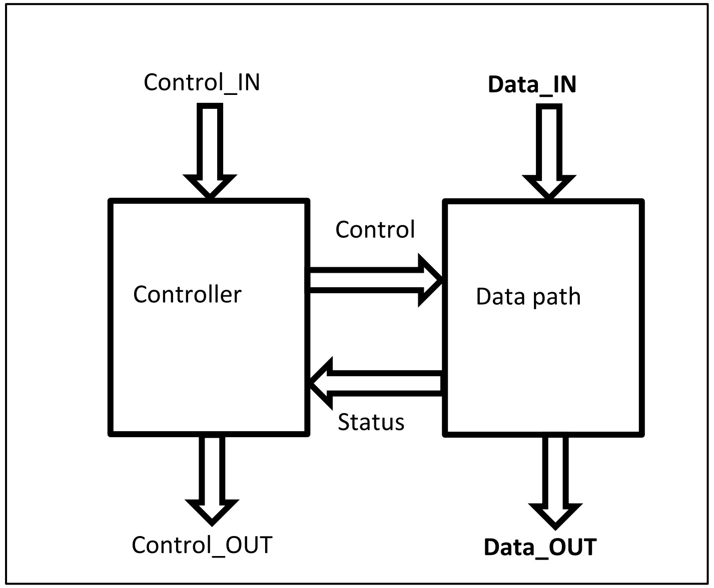
Many names:

- High-level Hardware synthesis
- **High-level synthesis (HLS)**
- Algorithmic synthesis
- Behavioral synthesis
- C Synthesis
- …

Classes of Hardware Components

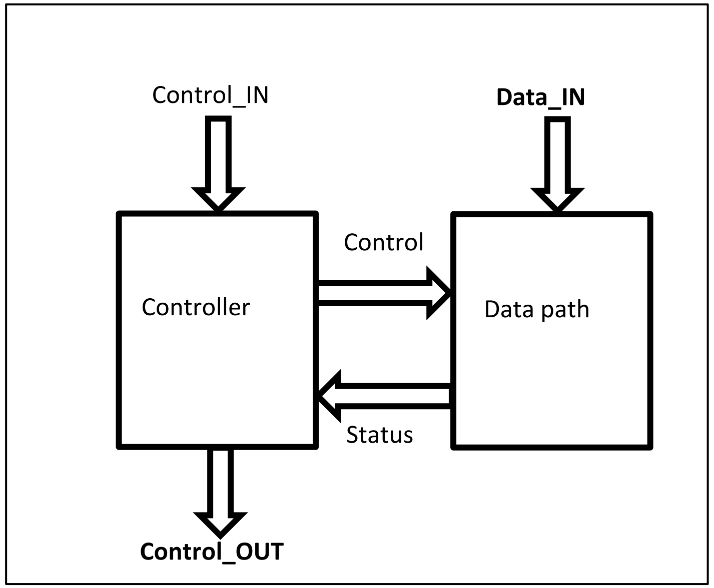
HLS works better on data-oriented designs

- Data-oriented designs



Examples: Video signal processing, compression, encryption,…

- Control-oriented designs



Examples: Traffic light control, industrial machines control, …

Performance metrics (1/2)

- **Clock Cycle Time** ΔT
  - Cycle duration of the driving clock of the HW module
  - The combinatorial path in the circuit with largest delay places an lower limit on the clock cycle time (critical path).
- **Latency** Λ
  - Number of clock cycles between the start of processing a block of data and the point of time at which the result is ready at the output.
- **Processing time**

$$t_{exe} = \Lambda \cdot \Delta T$$

- **Throughput** T
  - Number of blocks of data that can be processed in a fixed time.

Performance metrics (2/2)

- **Chip area (ASIC)**
  - Estimated via gate count.
  - Data path: Number of Hardware Operation Units such as multipliers, ALUs, registers, multiplexers,…
- **FPGA Resources**
  - Number of Luts, Number of DSP Blocks, …
- **Power/Energy Consumption**
  - Dynamic power consumption: Power consumed by switching transistors in the circuit
  - Static power consumption: Power consumed due to leakage currents.

Design Goals and Constraints

- Synthesis algorithms handle two typical cases:

- **Timing constrained:**
  - Constrained: Implement task such that it can compute result in maximal number of clock cycles (maximal latency).
  - Goal: Minimize number of functional units (adders, ALUs, multipliers) in data path.
  - Second goal: Minimize number of registers (register sharing), multiplexers, control unit states, …

- **Resource constrained:**
  - Constrained: Implement task with fixed maximal number of functional units (adders, ALUs, multipliers) in data path.
  - Goal: Minimize latency.
  - Second goal: Minimize number of registers (register sharing), multiplexers, control unit states, …

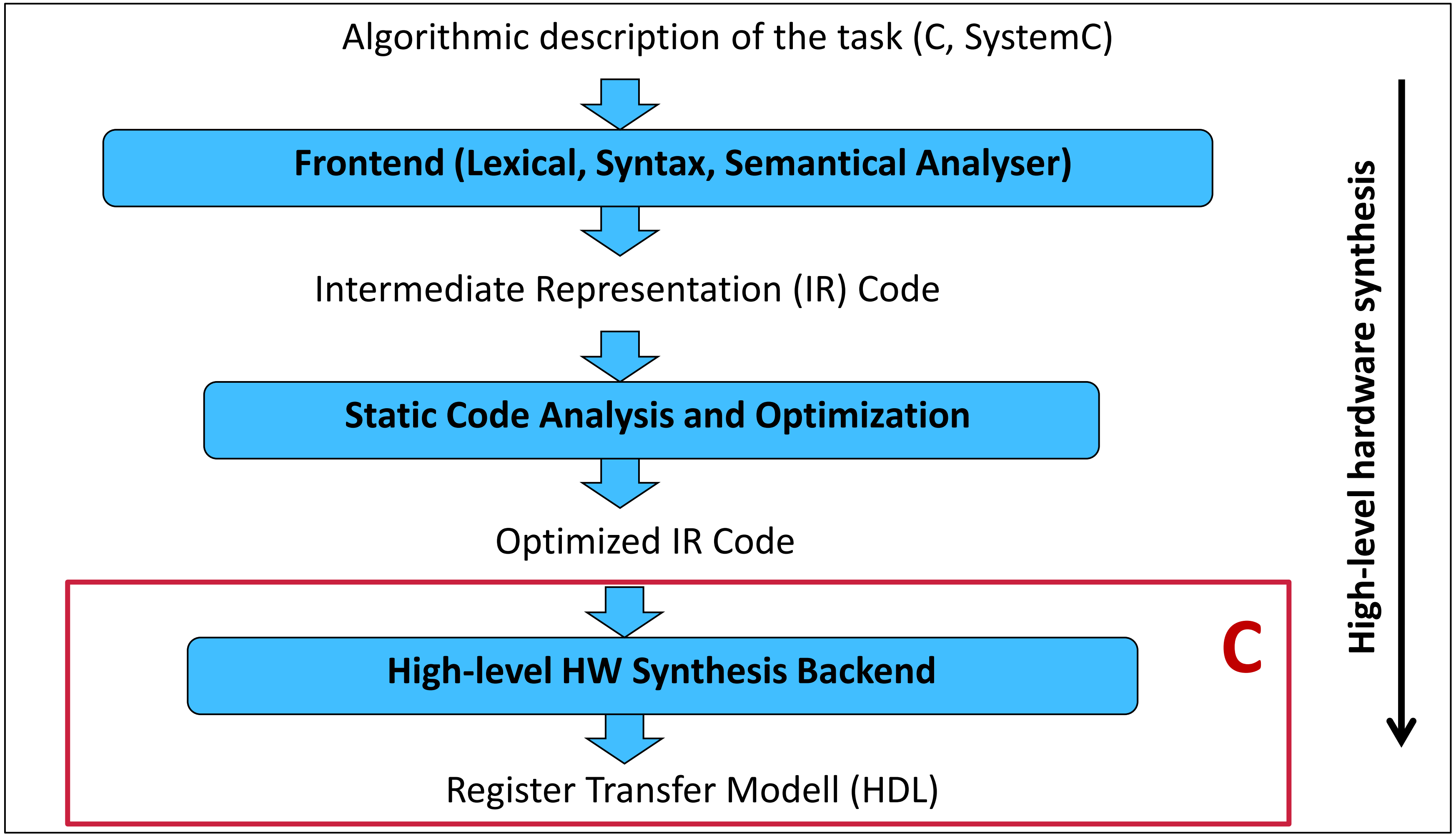
Synchronous HW Design

- All registers in control unit and data path share same clock.

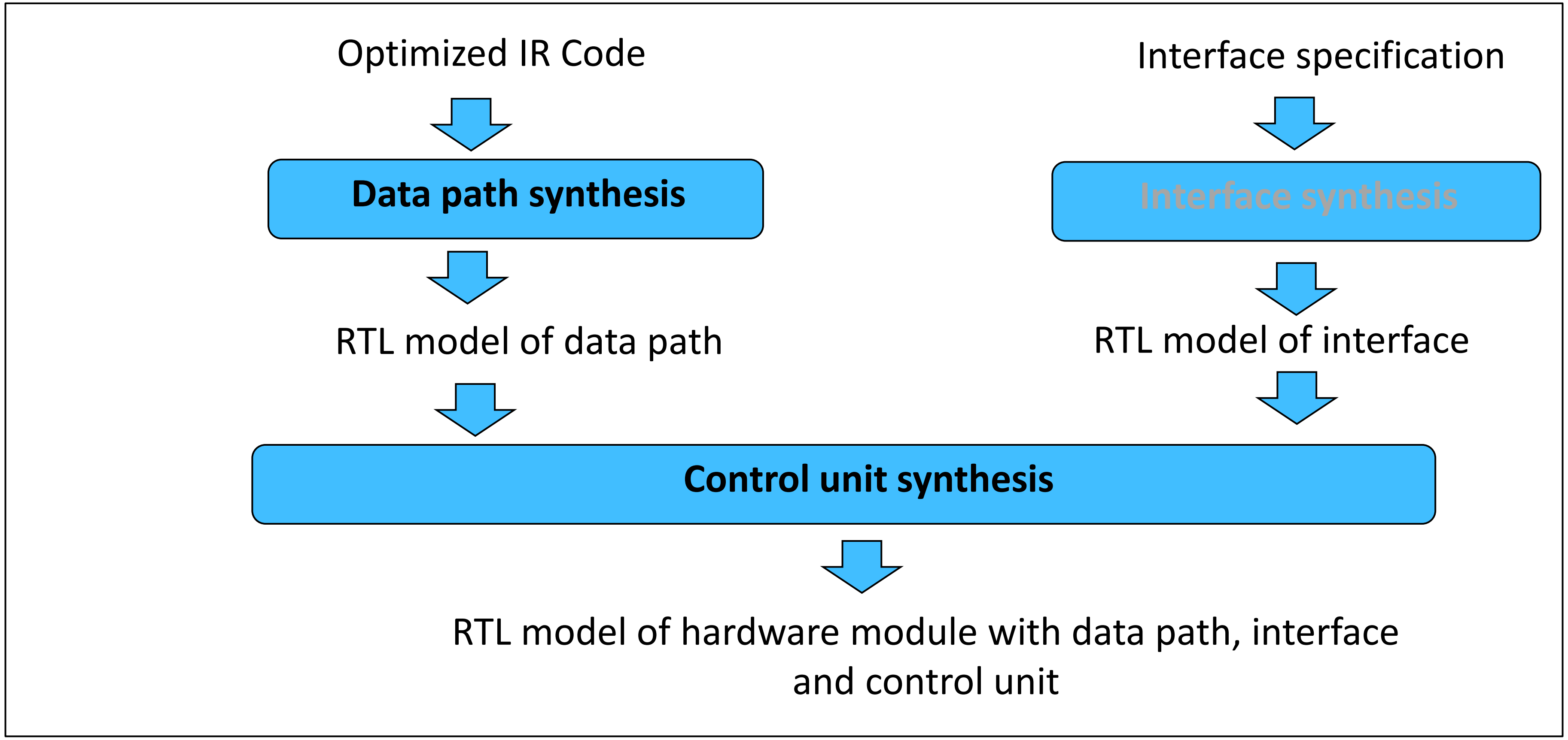
- Assumptions for simplification:
  - Functional units have a fixed and known delay such that the number of clock cycles to execute operation is assumed to be fixed and data-independent.
  - The delay of interconnect and multiplexers can be neglected.

- Real life:
  - Longest combinatorial path in the circuit will determine the maximal clock frequency.
  - Logic synthesis will try to optimize circuit dependent on target clock frequency and area.

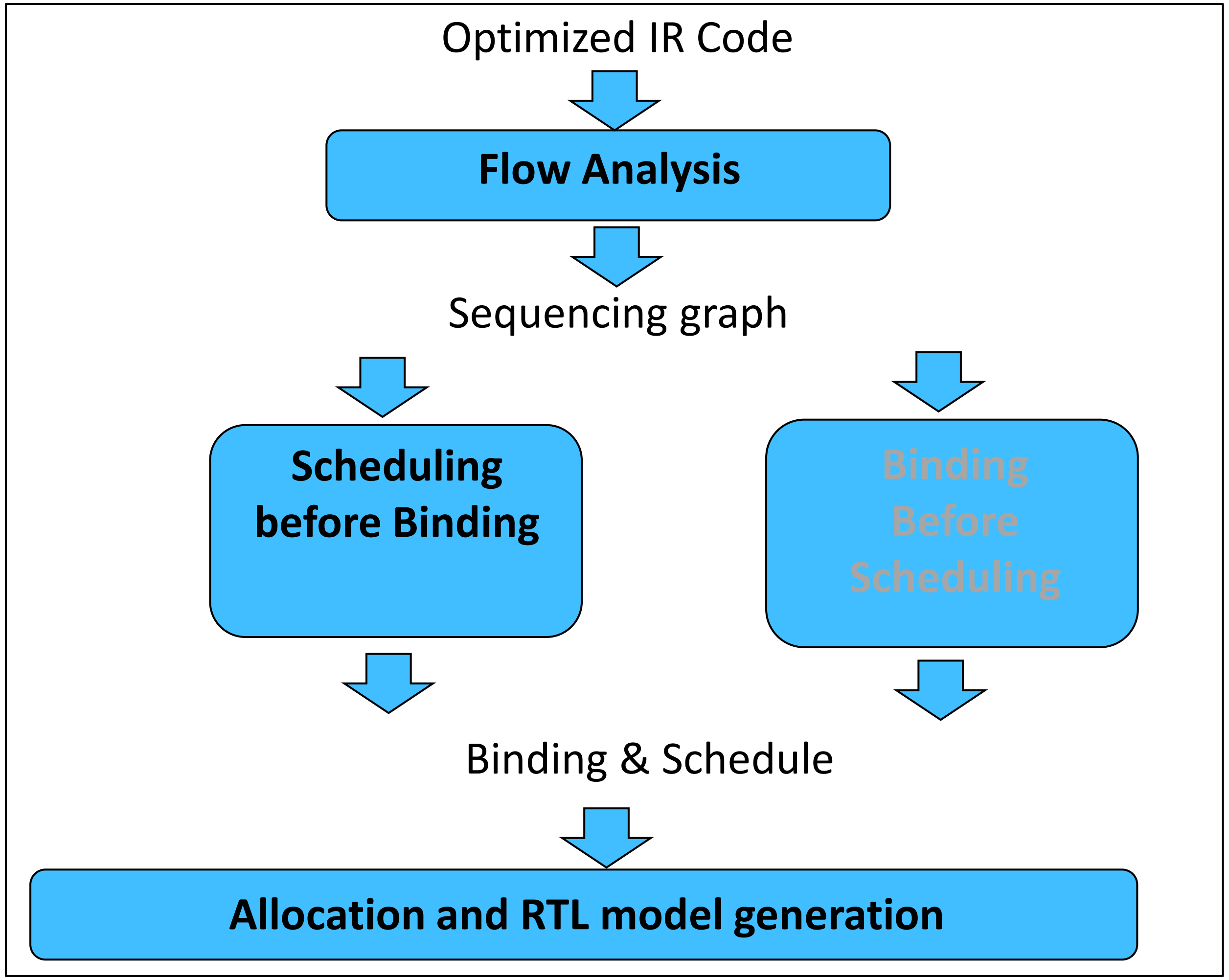
High-level HW Synthesis Flow



High-level HW Synthesis Backend



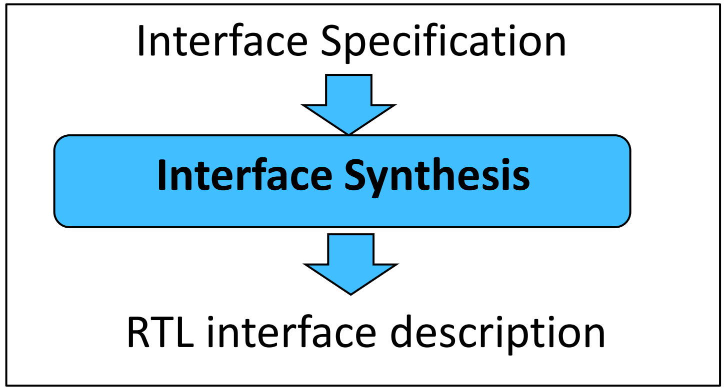
C1.8. Data Path Synthesis



Data Path Synthesis Steps

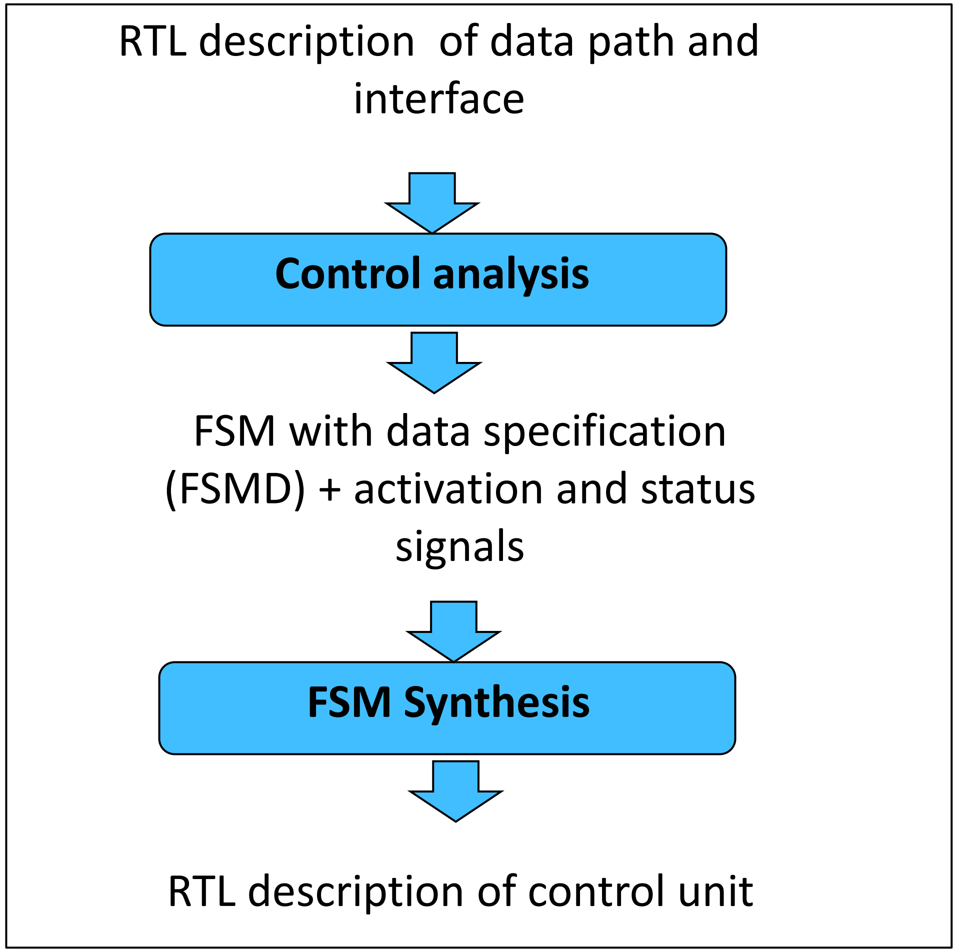
- **Scheduling**:
  - Determines the start time of each operation
- **Binding:**
  - Determine on which functional units the operation is executed.
  - Determine in which register variables are saved.
- **Allocation**:
  - Selection of resources such as functional units, registers and multiplexers.

Interface Synthesis



- Interfaces can differ strongly.
- Interface may consist of:
  - Memory, Registers or FIFOS as data buffers.
  - FSMs for communication (bus) protocols.
- Crossing of clock domains possible, e.g. between bus clock and HW module clock.

Control Unit Synthesis



D1-3 Data Path Synthesis
HW Resources

- Literature:
- „*Specification and Design of Embedded Systems*“ Daniel D. Gajski, Prentice Hall 1994
- „*Digitale Hardware/Software Systeme*“, Jürgen Teich, Springer 1997
- „*Embedded System Design*“, Daniel D. Gajski et.al., Springer 2009

HW Resources in the Data Path

- Functional units: Adder, Multiplier, ALUs,…
  - Execute operations on data, e.g., Add, Shift, AND, OR, Mult, …
  - Fixed and known delay
  - Fixed and known area demand
- Signal nets and multiplexers
  - Delay and area demand is neglected.
- Memory elements: Registers
  - Delay and area demand is neglected.
- NFU (Nonfunctional Unit):
  - Nonexistent helper resource
  - used to execute special NOP, LOOP, BRANCH, CALL operations (more later)

HW Resources in the Data Path

- Functional units are identified by a pair

$$(k_r, z_r)$$

  - of their type:

$$k_r \in K$$

  - and an index

$$K = \{\mathrm{ALU}, \mathrm{MULT}, ...\}$$

$$z_r = 1, 2, ...$$

  - Example:

$$(ALU, 1), (ALU, 2), (MULT, 1)$$

Time-Resource-Plane (TRP) (1/4)

- X-axis: Resources
  - List allocated operational units
  - Assign operations to operational units (Binding)

- y-axis: Time
  - Division in clock cycles.
  - Plan temporal order of the operations
  - Select start times of operations (Schedule)
  - Values must be saved in registers between clock cycles.

Time-Resource-Plane (TRP) (2/4)

- Example: Goertzel Algorithm

  (Basic block B3)

```
t6= s_prev1 * s_prev1
t7= s_prev2 * s_prev2
t8= s_prev1 * s_prev2
t9= t8 * coeff
t10= t6+t7
power= t10 - t9
```

Resources (Functional units) →

	Add,1	Mult,1	Mult,2
CC 1		t6= s_prev1* s_prev1	t8= s_prev1* s_prev2
CC 2		t7= s_prev2* s_prev2	t9= t8* coeff
CC 3	t10= t6+t7		
CC 4	power= t10-t9		

↓ Time in clock cycles (CC)

- Example: Goertzel Algorithm

(Basic block B3)

```
t6= s_prev1 * s_prev1
t7= s_prev2 * s_prev2
t8= s_prev1 * s_prev2
t9= t8 * coeff
t10= t6+t7
power= t10 - t9
```

	Add,1	Mult,1
CC 1		t6= s_prev1* s_prev1
CC 2		t7= s_prev2* s_prev2
CC 3	t10= t6+t7	t8= s_prev1* s_prev2
CC 4		t9= t8* coeff
CC 5	power= t10-t9	

Time-Resource-Plane (TRP) (4/4)

- Example: Goertzel Algorithm

(Basic block B3)

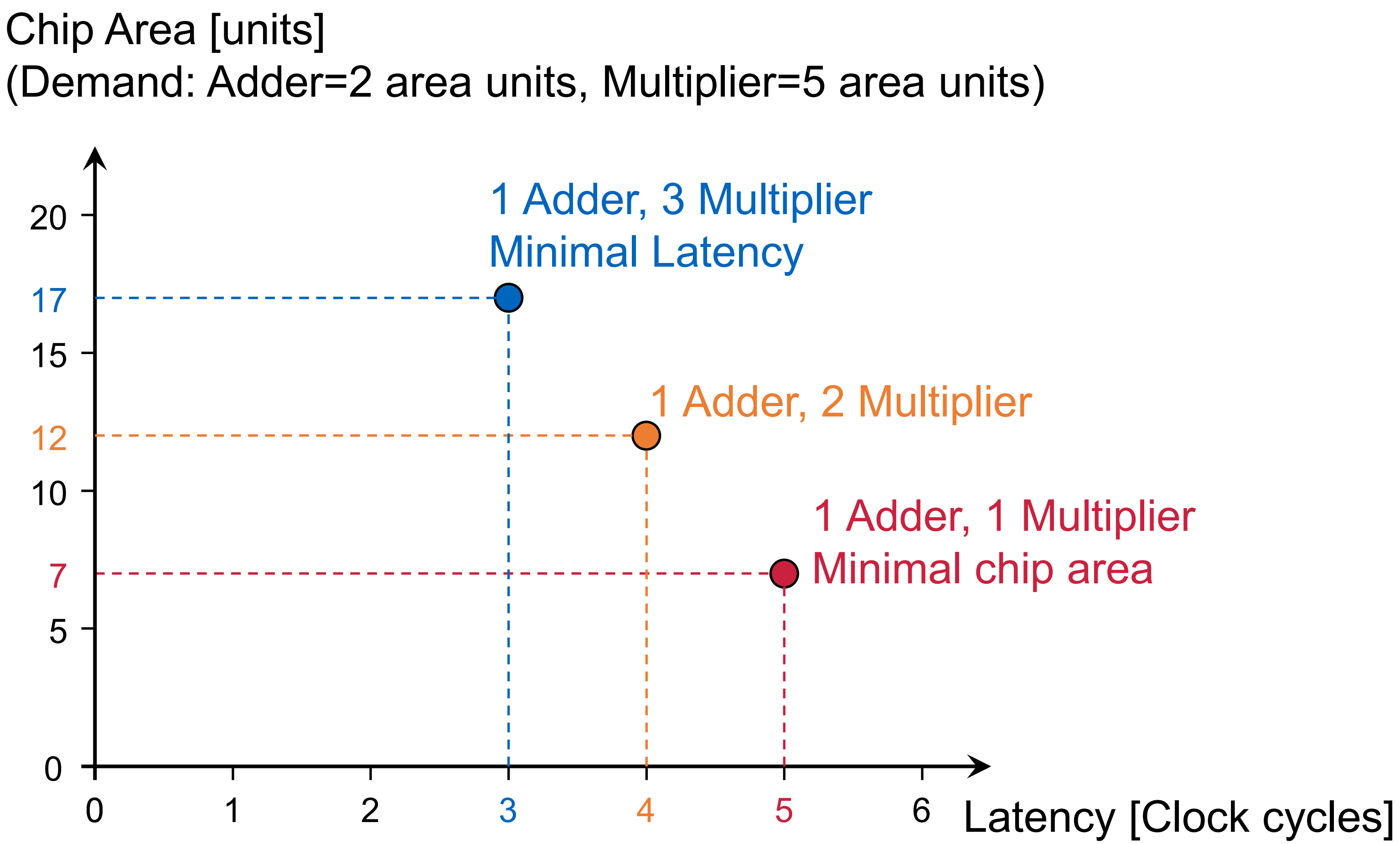
```
t6= s_prev1 * s_prev1
t7= s_prev2 * s_prev2
t8= s_prev1 * s_prev2
t9= t8 * coeff
t10= t6+t7
power= t10 - t9
```

	Add,1	Mult,1	Mult,2	Mult,3
CC 1		t6= s_prev1* s_prev1	t7= s_prev2* s_prev2	t8= s_prev1* s_prev2
CC 2	t10= t6+t7	t9= t8* coeff		
CC 3	power= t10-t9			

Pareto-Optimality (1/2)

- Which is the best solution?
- Solution is Pareto optimal, if there exist no solution that is better in all design performance metrics.
- Different Pareto-optimal solutions allow different trade-offs between the design performance metrics.
- Best solution is picked based on preferences on design performance metrics.

Pareto-Optimality (2/2)



D1-4 Sequencing Graphs

Sequencing Graph (SG)

- Sequencing graph: $G_s(V_s, E_s)$

- Hierarchy of directed acyclic graphs (DAGs). Each DAG is called a sequencing graph unit (SGU).
- SGUs are polar: One source and one sink node is added, which is labeled No operation (NOP).

- Nodes: $V_s = \{v_i : i = 0, ..., n\}$
  - No operation (NOP)
  - Operations (+,>,<,*,…)
  - Hierarchical node (CALL, BR, LOOP)

- Edges: $E_s = \{(v_i, v_j) : i, j = 0, ..., n\}$
  - between nodes in one SGU unit: Data dependency between two operations
  - between Source/sink and hierarchical nodes: Connections between SGU on different hierarchical levels
- Paths describe concurrent operations that may possibly be executed in parallel,

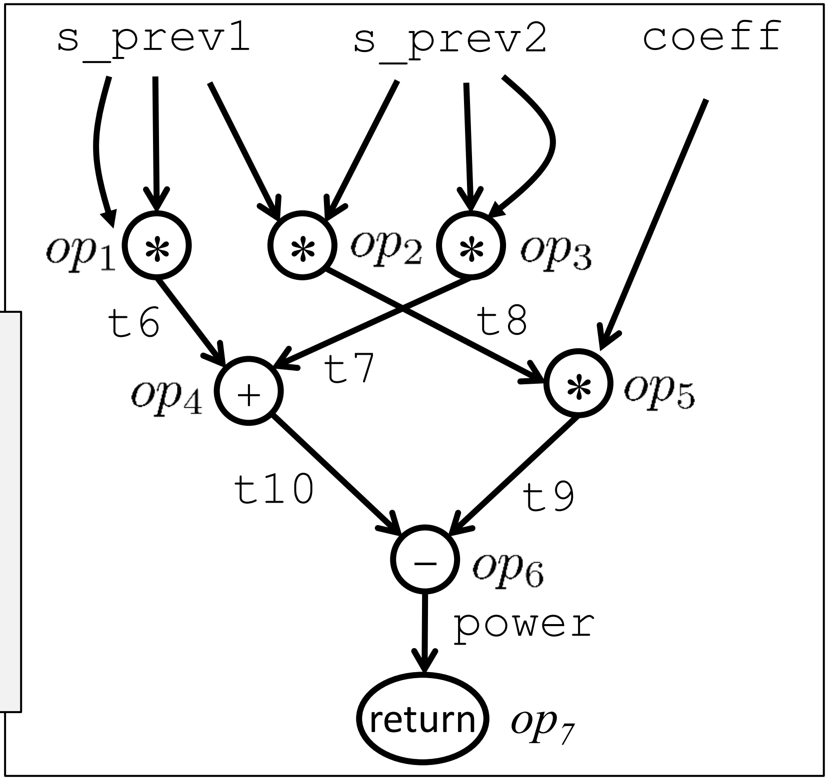
- Example: SGU for basic block B3 of Goertzel algorithm

- Example: Goertzel Algorithm

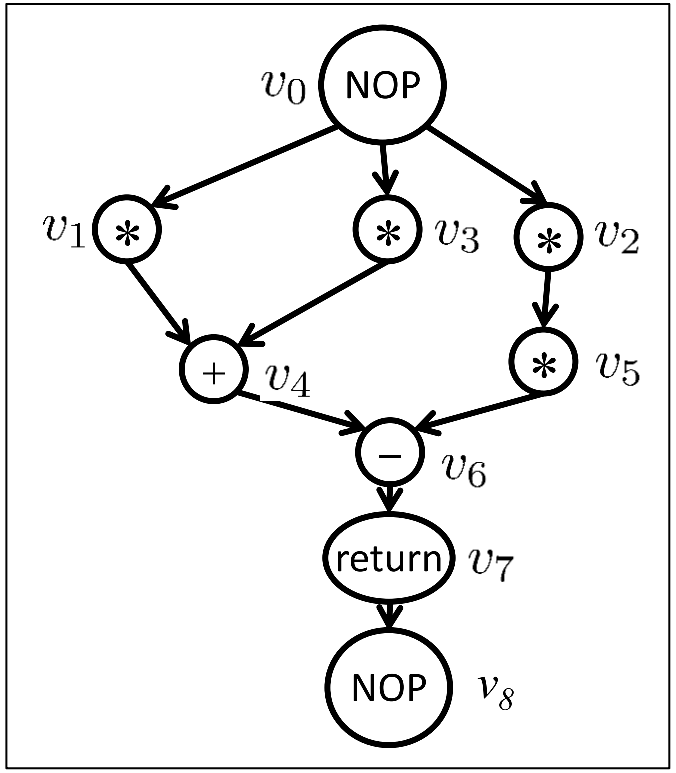
(Basic block B3)

```
t6= s_prev1 * s_prev1
t7= s_prev2 * s_prev2
t8= s_prev1 * s_prev2
t9= t8 * coeff
t10= t6+t7
power= t10 - t9
```

Data flow graph



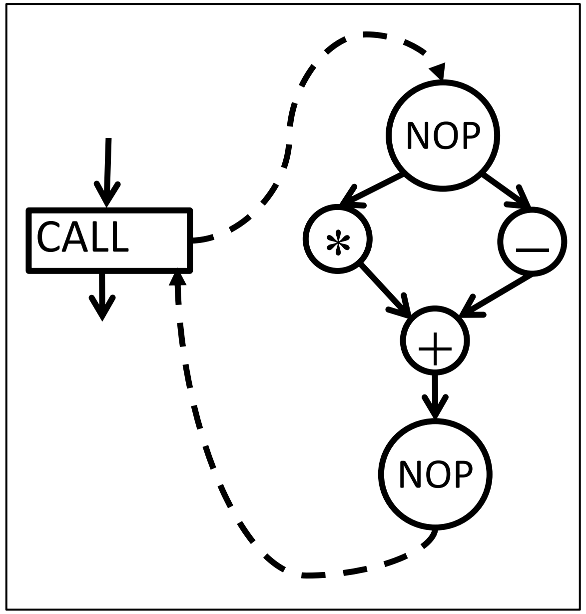
Sequencing graph unit



Sequencing Graph

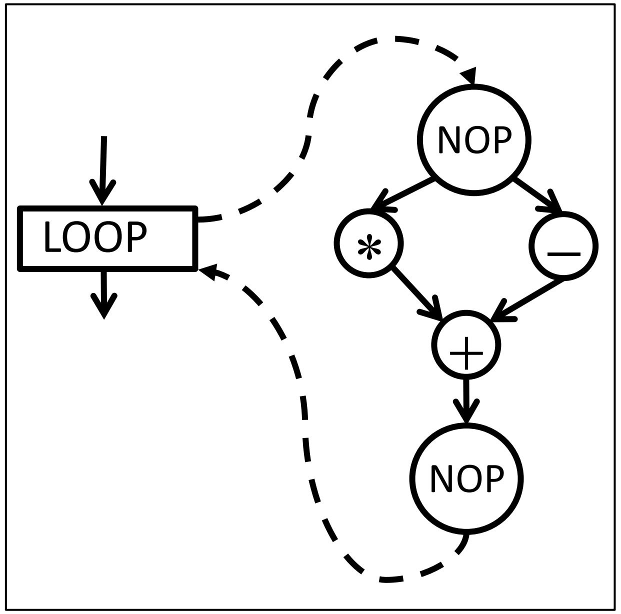
- Hierarchical nodes: CALL, LOOP, BR

Call to procedure



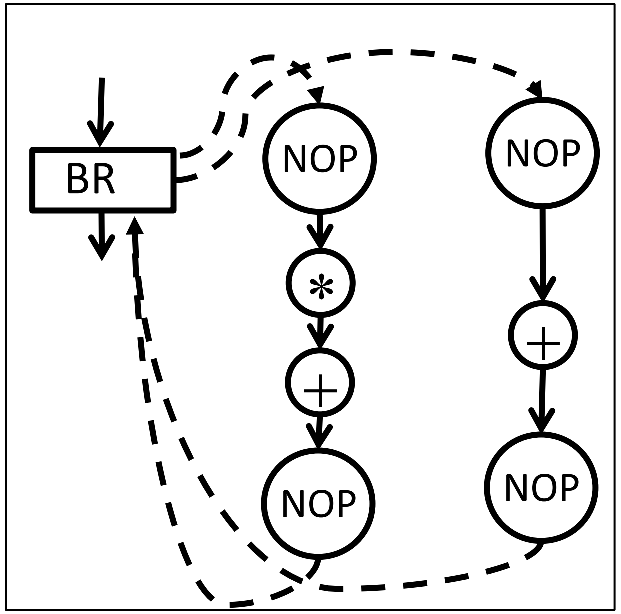
Called SGU of one lower hierarchical level is executed once.

Control flow loop



SGU of lower hierarchical level is executed 0 to N times.

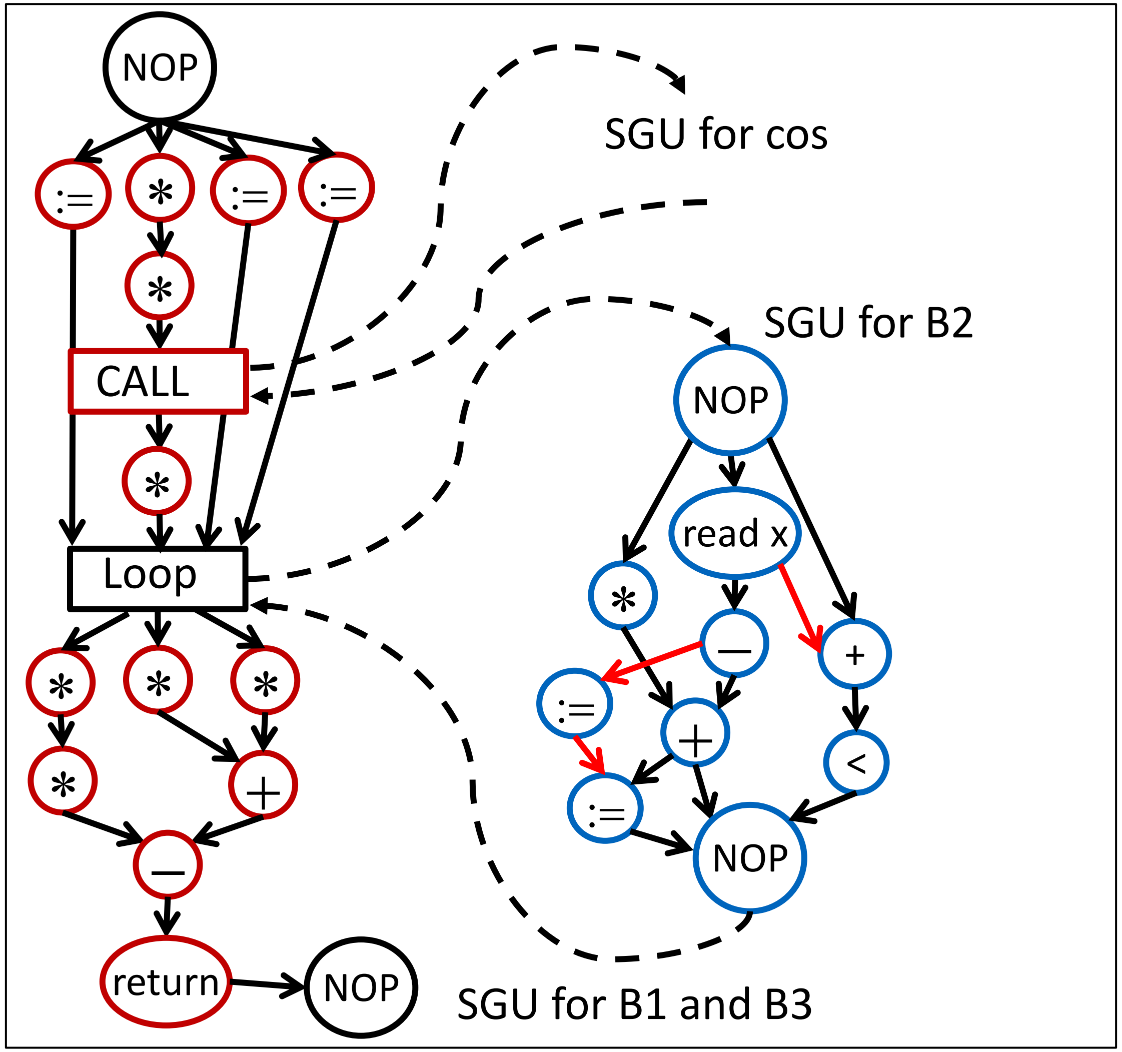
Control flow branch



Only one of the two SGU of lower hierarchical level is executed once.

Sequencing Graph

- Example: Goertzel algorithm




```
B1: s_prev1 := 0.0
    s_prev2 := 0.0
    i:=0
    t1 := 2*3.14
    f := t1 * freq
    param f
    t2 := call cos,1
    coeff:=2.0*t2
```

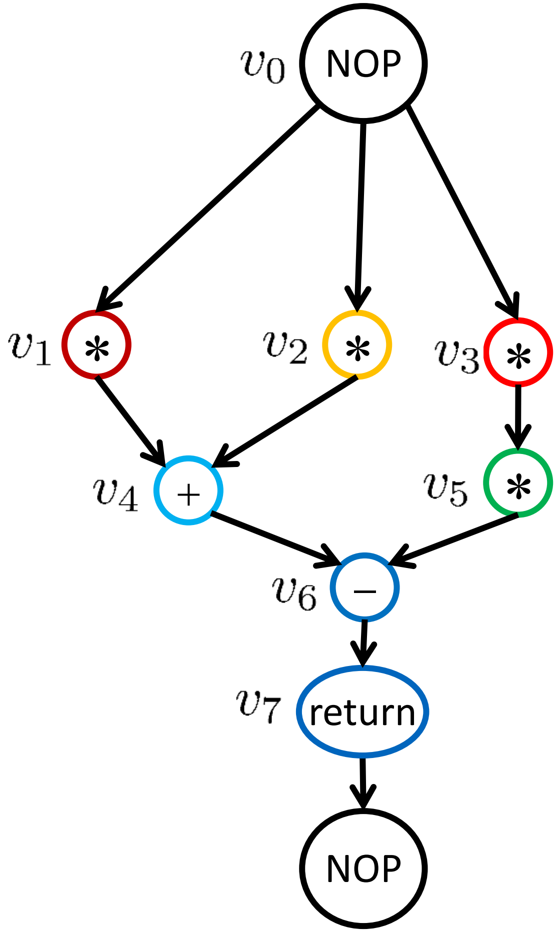
```
B2: t3:= coeff * s_prev1
    t4:= x[i]
    t5 := t4 - s_prev2
    s := t3 + t5
    s_prev2 := s_prev1
    s_prev1 := s
    i:=i+1
    if i < 64 goto B2
```

```
B3: t6:= s_prev1 * s_prev1
    t7:= s_prev2 * s_prev2
    t8:= s_prev1 * s_prev2
    t9:= t8 * coeff
    t10:= t6+t7
    power:= t10 - t9
    return power
```

- Example: Goertzel Algorithm

(Basic block B3)

```
t6= s_prev1 * s_prev1
t7= s_prev2 * s_prev2
t8= s_prev1 * s_prev2
t9= t8 * coeff
t10= t6+t7
power= t10 - t9
return power
```

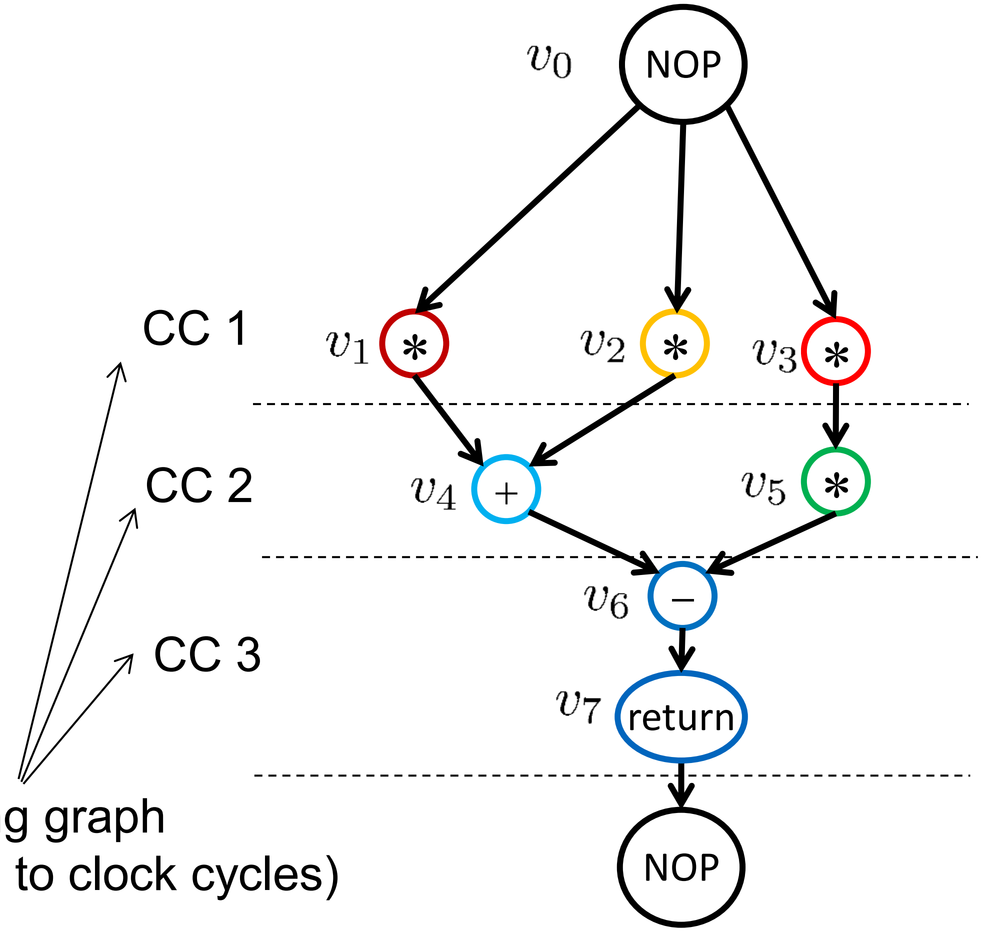


Sequencing Graph in the TRP

- Example: Goertzel Algorithm

  (Basic block B3)

```
t6= s_prev1 * s_prev1
t7= s_prev2 * s_prev2
t8= s_prev1 * s_prev2
t9= t8 * coeff
t10= t6+t7
power= t10 - t9
return power
```



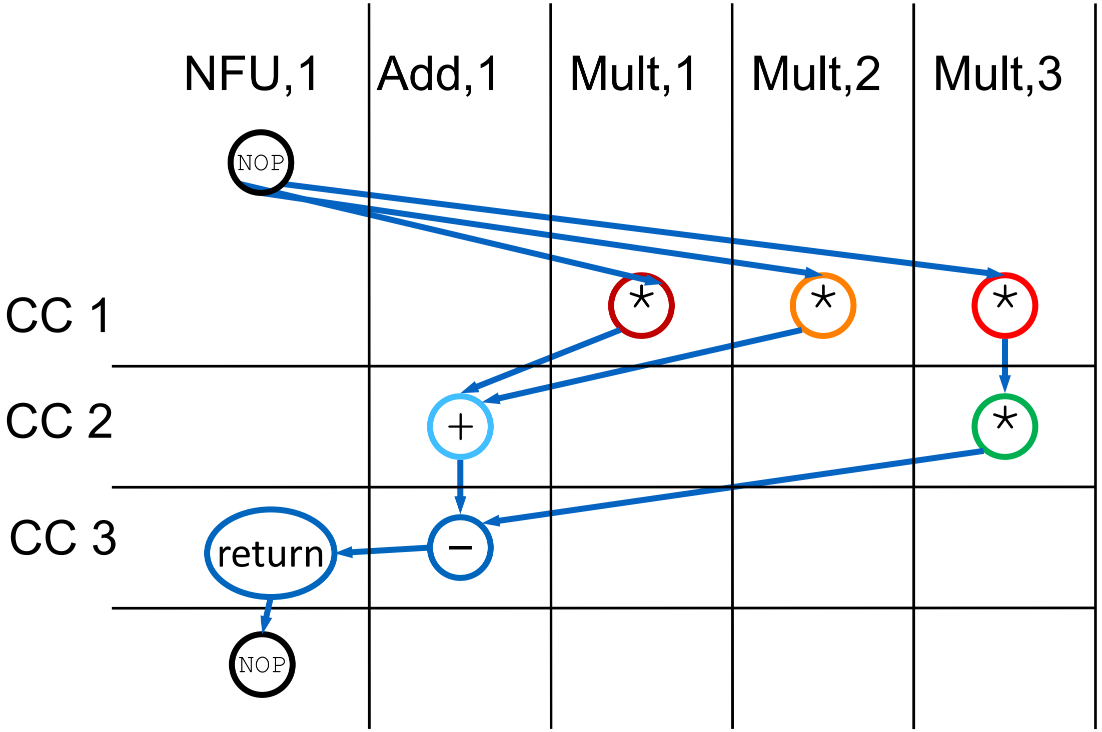
Scheduled sequencing graph
(Operations assigned to clock cycles)

Sequencing Graph in the TRP

- Example: Goertzel Algorithm

(Basic block B3)

```
t6= s_prev1 * s_prev1
t7= s_prev2 * s_prev2
t8= s_prev1 * s_prev2
t9= t8 * coeff
t10= t6+t7
power= t10 - t9
return power
```

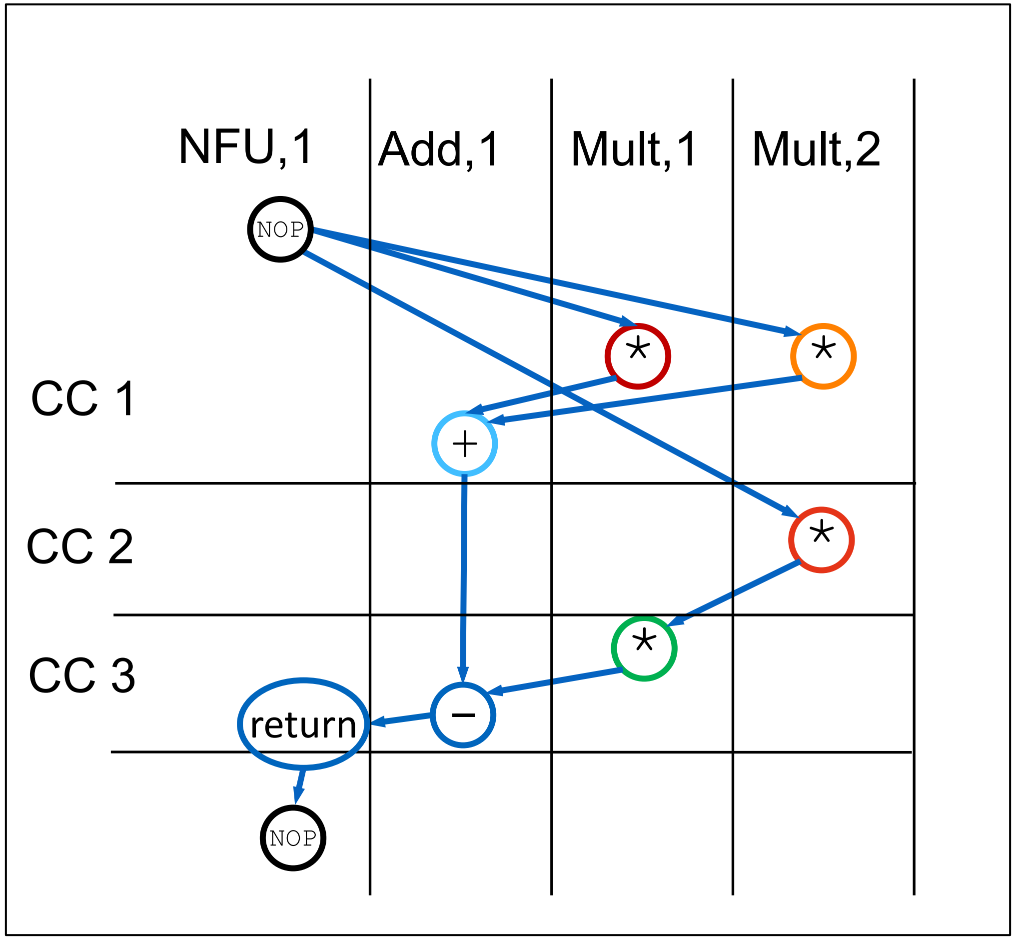


Scheduled sequencing graph with binding
(Operations assigned to clock cycles and operational units)

Operation Chaining

- Delay of operational units allow two operations in one clock cycle.

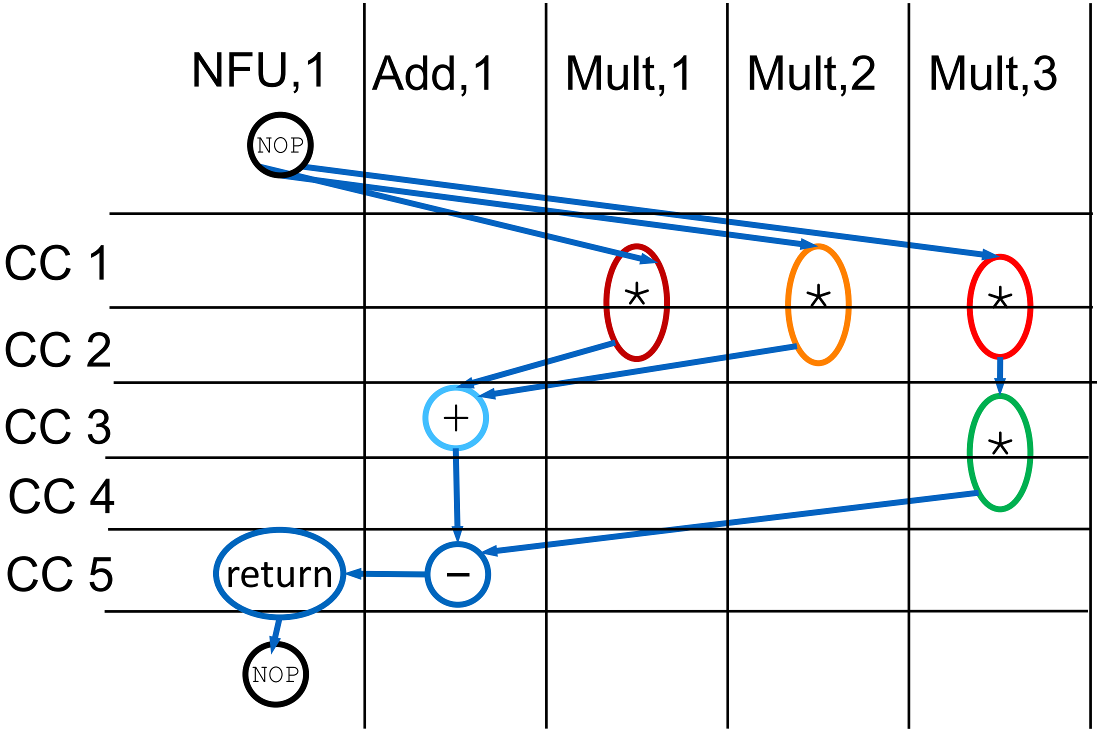
- Example: Goertzel Algorithm
  - Addition and multiplication in same clock cycle possible.
  - Operational units must be switched in series (chained).



Multi-cycle Operations

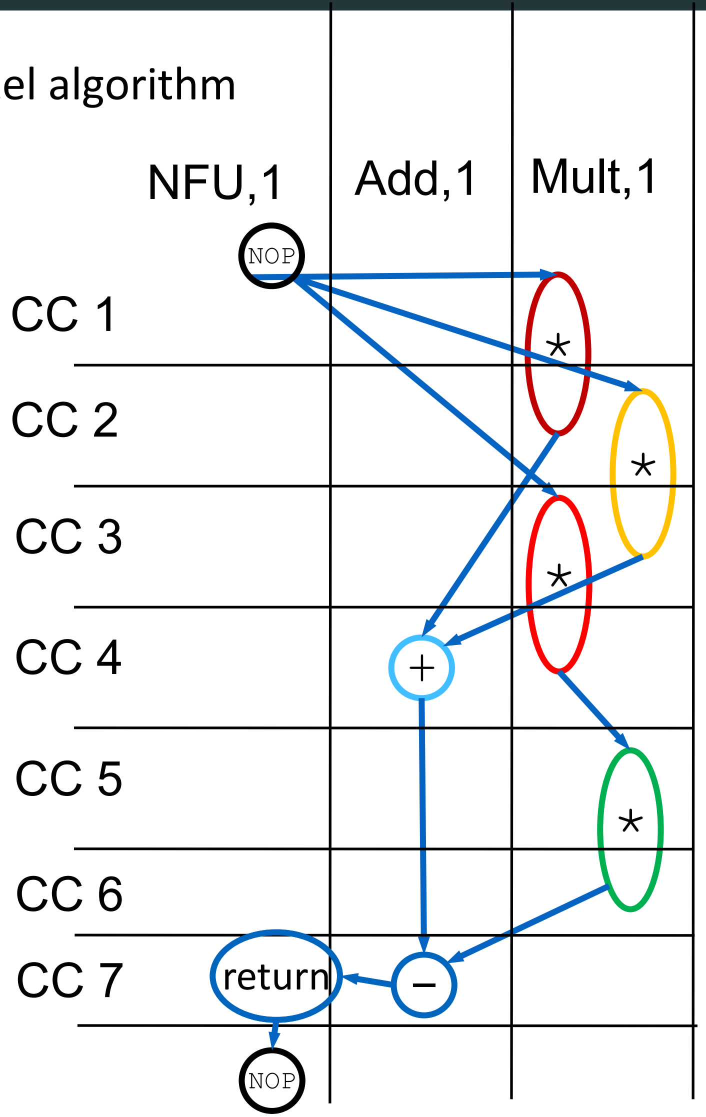
- Delay of functional elements requires several clock cycles for the execution of the operation.

- Example:
  Goertzel algorithm



Pipelined Operational Units

- Example: Goertzel algorithm



- New operation can start before previous operation has finished.
- Number of concurrent operations is equal to pipeline depth.
- Operational units has internal registers to save intermediate values.

Summary

Where we are

- HLS is a step that uses C specification to design an HW IP block
- This HW IP block executes the algorithm specified in C