High Level Synthesis

- Data Flow Graphs
- FSM with Data Path
- Allocation
- Scheduling
- Implementation
- Directions in Architectural Synthesis

Source: Jacob Abraham
University of Texas at Austin

High Level Synthesis (HLS)

• Convert a high-level description of a design to a RTL netlist

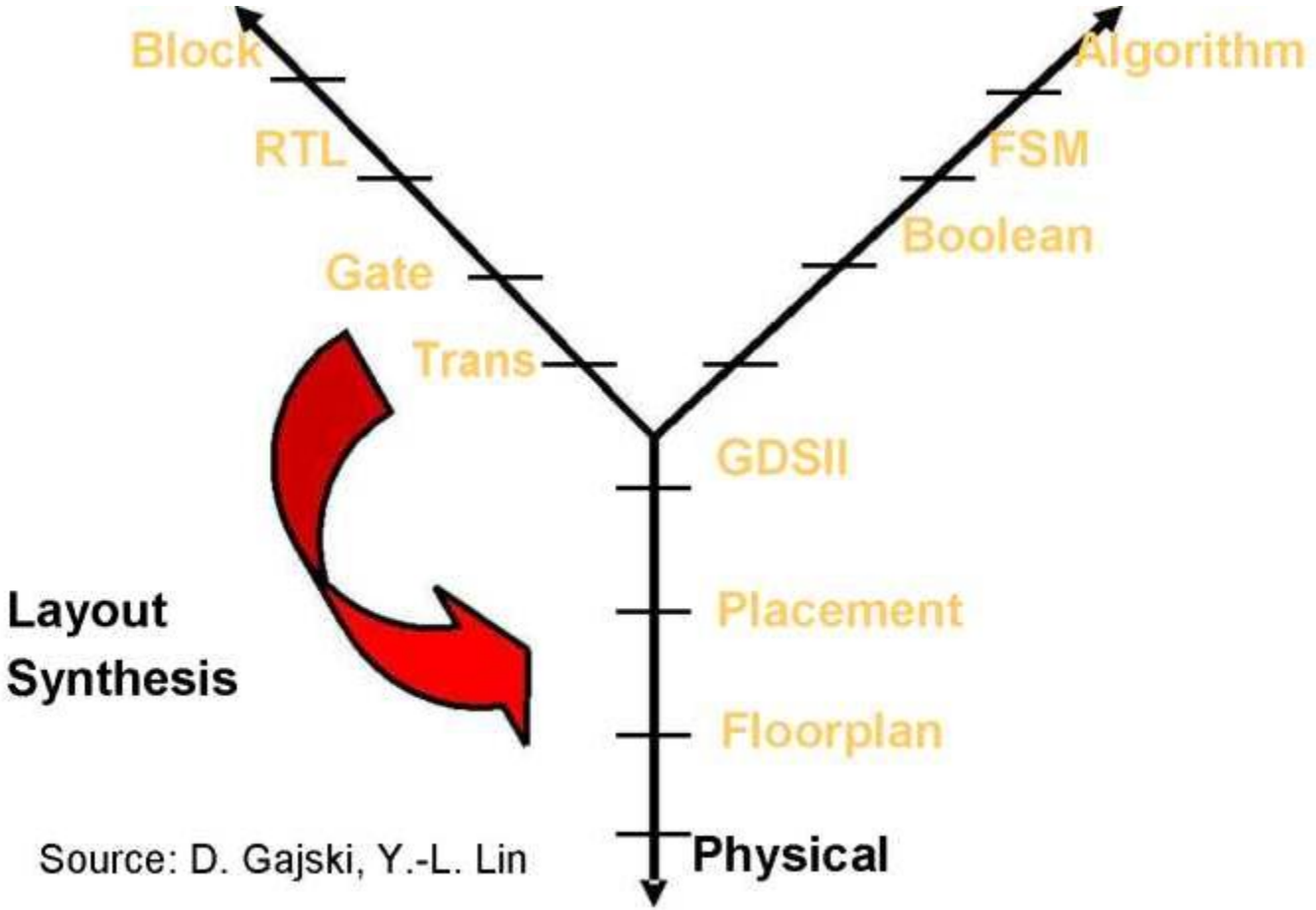
- Input:
  - High-level languages (e.g., C)
  - Behavioral hardware description languages (e.g., VHDL)
  - State diagrams / logic networks

- Tools:
  - Parser
  - Library of modules

- Constraints:
  - Area constraints (e.g., # modules of a certain type)
  - Delay constraints (e.g., set of operations should finish in X clock cycles)

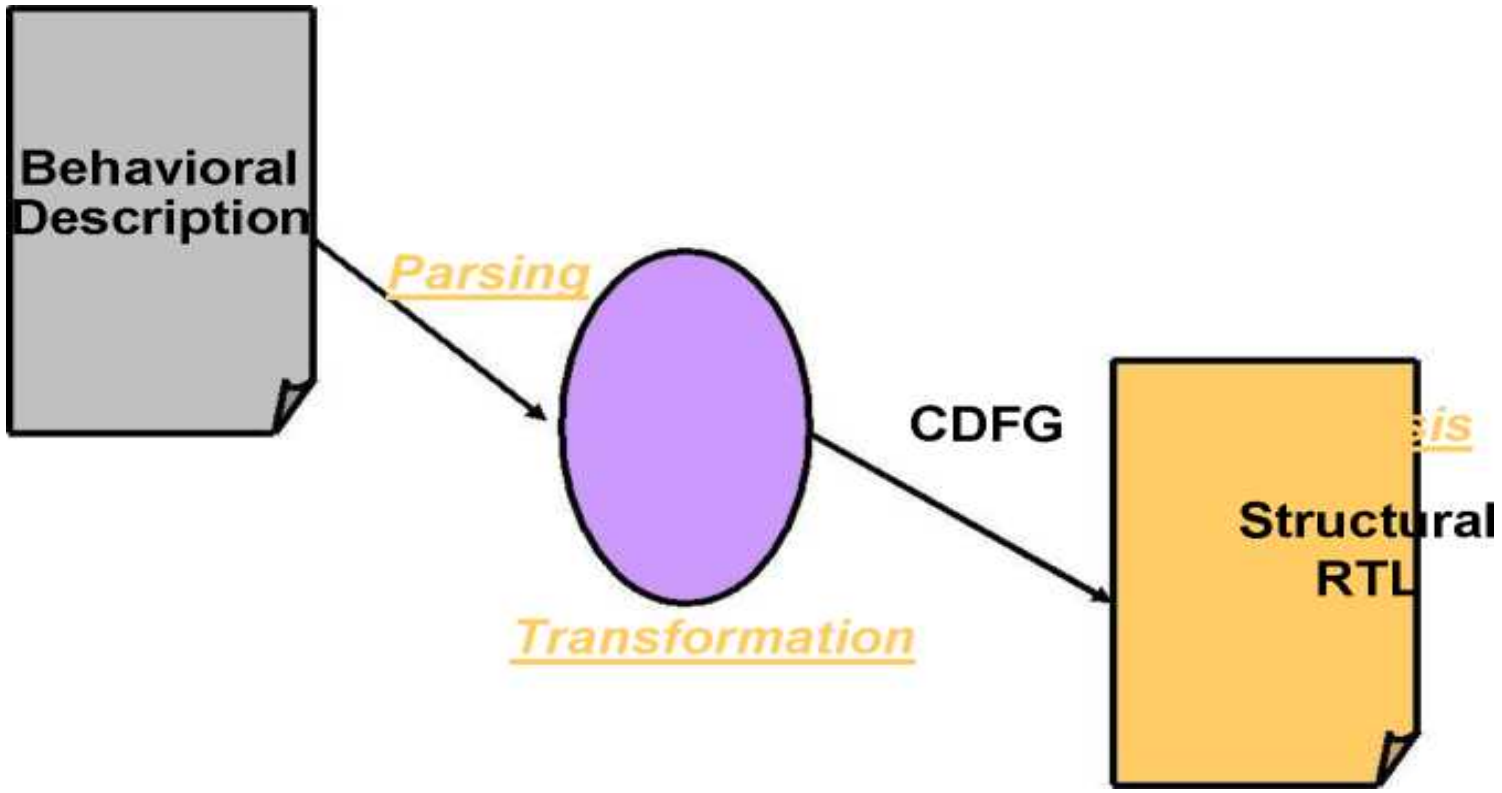
- Output:
  - Operation scheduling (time) and binding (resource)
  - Control generation and detailed interconnections

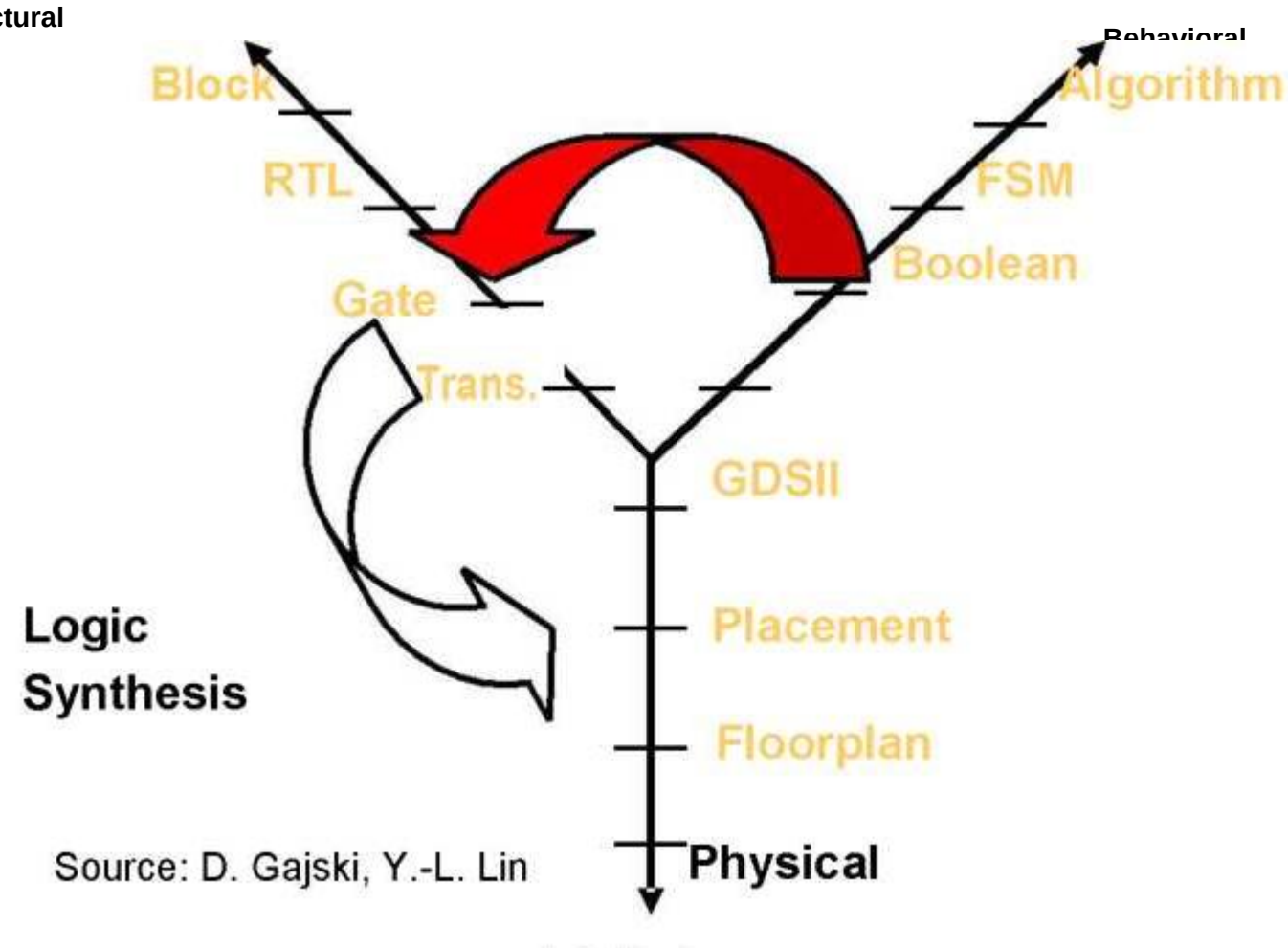
Structural

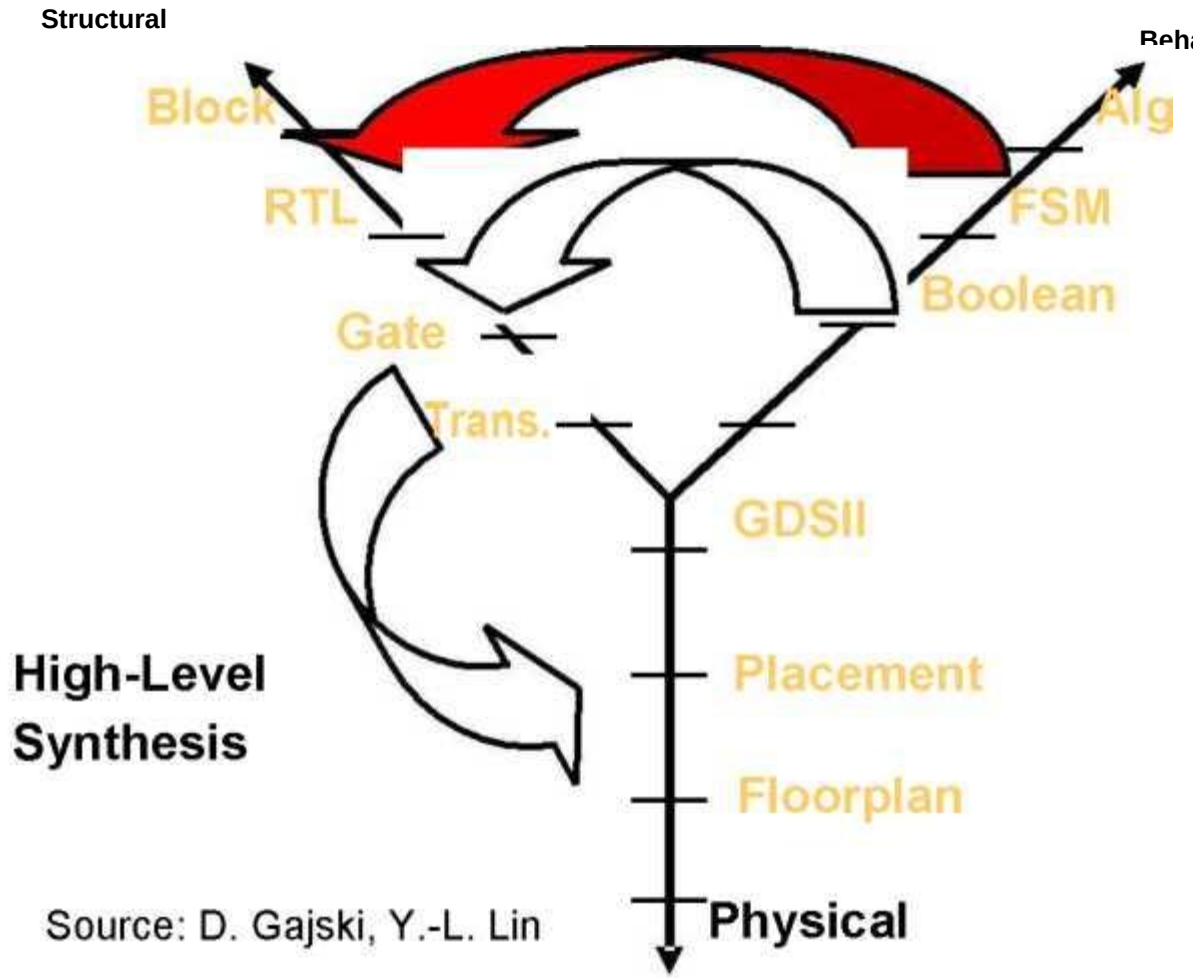


Behavioral

High Level Synthesis




Structural
Behavioral
Block
Algorithm
RTL
FSM
Boolean
Gate
Trans.
GDSII
Placement
Floorplan
Physical
Logic
Synthesis
Source: D. Gajski, Y.-L. Lin


Structural
Behavioral
Block
Alg
RTL
FSM
Boolean
Gate
Trans.
GDSII
Placement
Floorplan
Physical
High-Level
Synthesis
Source: D. Gajski, Y.-L. Lin

Essential Issues

- Behavioral Specification Languages
- Target Architectures
- Intermediate Representation
- Operation Scheduling
- Allocation/Binding
- Control Generation

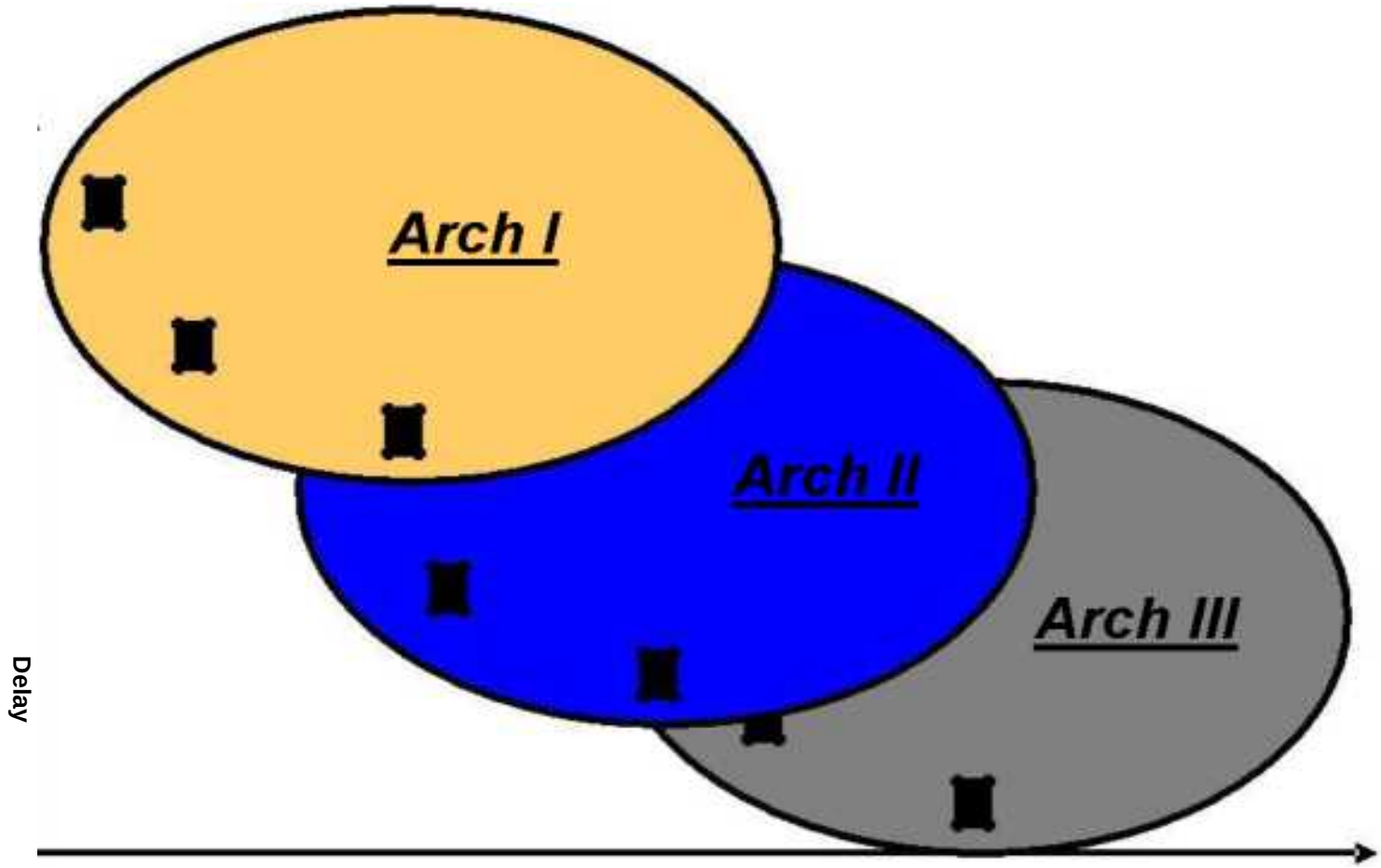
Behavioral Specification Languages

- Add hardware-specific constructs to existing languages
  - SystemC
- Popular HDL
  - Verilog, VHDL
- High level programming languages
  - C, Java, Python

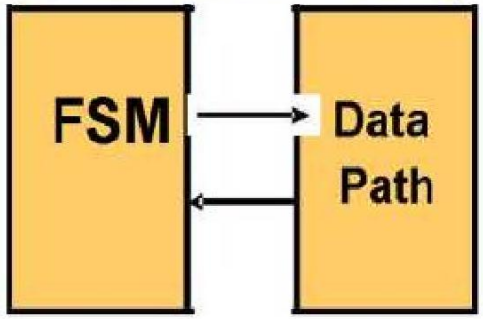
Target Architectures

- Bus-based
- Multiplexer-based
- Register file
- Pipelined
- RISC, VLIW
- Interface Protocol

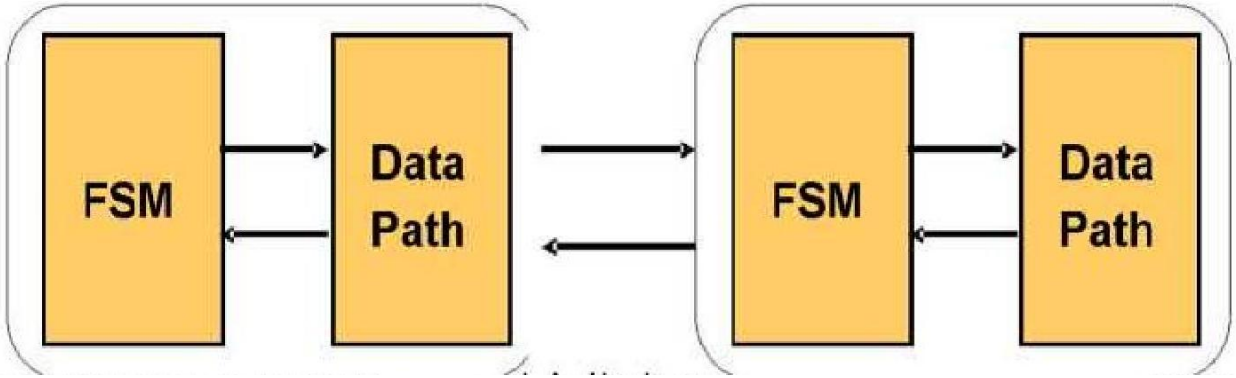
Design Space Exploration



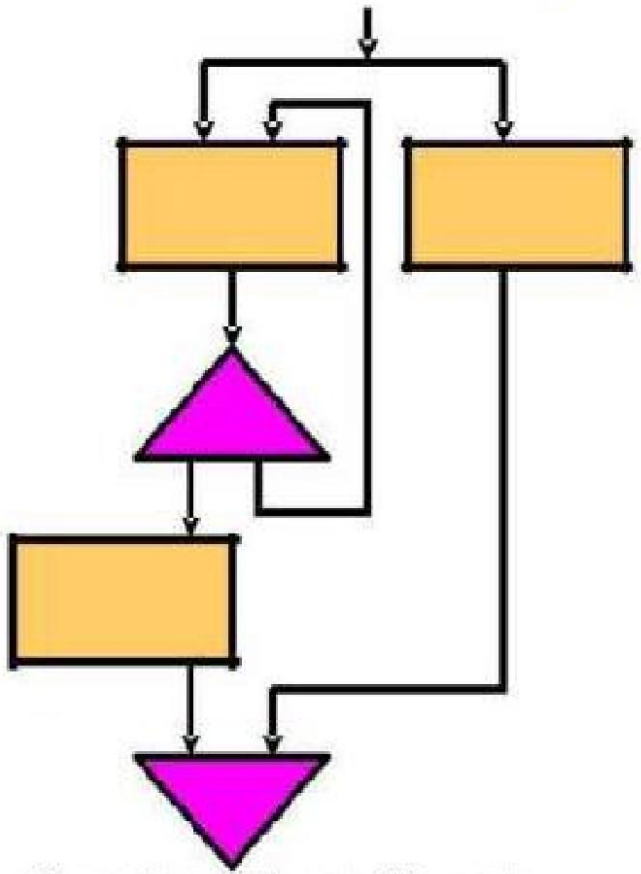
FSM with Data Path (FSMD)



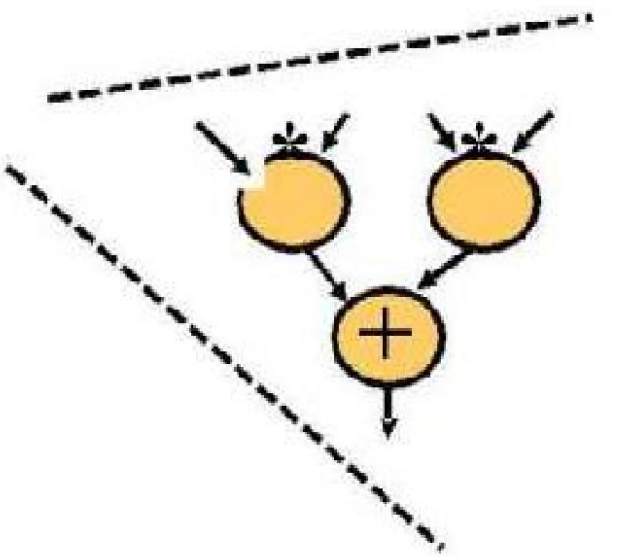
Communicating FSMDs



Intermediate Representation (CDFG)



Control Flow Graph



Data Flow Graph

Quality Measures for High-Level Synthesis

- Performance
- Area Cost
- Power Consumption
- Testability
- Reusability

Hardware Variations

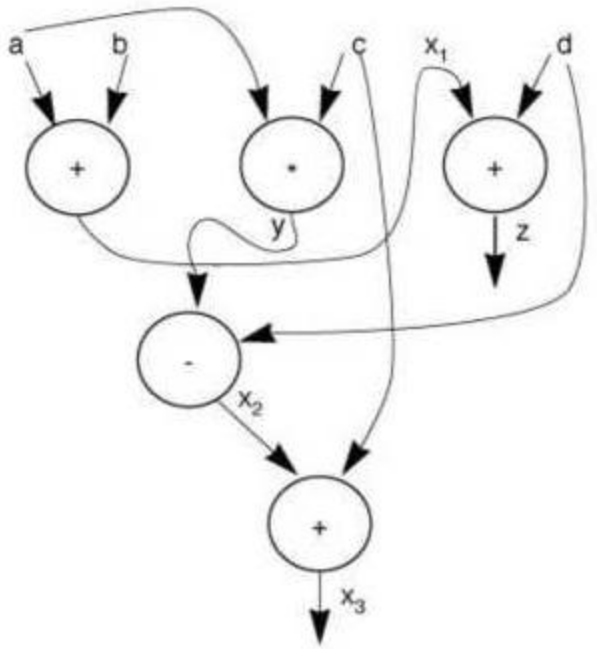
- Functional Units
  - Pipelined, Multi-Cycle, Chained, Multi-Function
- Storage
  - Register, RF, Multi-Ported, RAM, ROM, FIFO, Distributed
- Interconnect
  - Bus, Segmented Bus, Mux, Protocol-Based

Data flow graph

- Data flow graph (DFG) models data dependencies
- Does not require that operations be performed in a particular order
- Models operations in a basic block of a functional model—no conditionals
- Requires single-assignment form

Data flow graph construction, cont'd

Data flow forms directed acyclic graph (DAG):

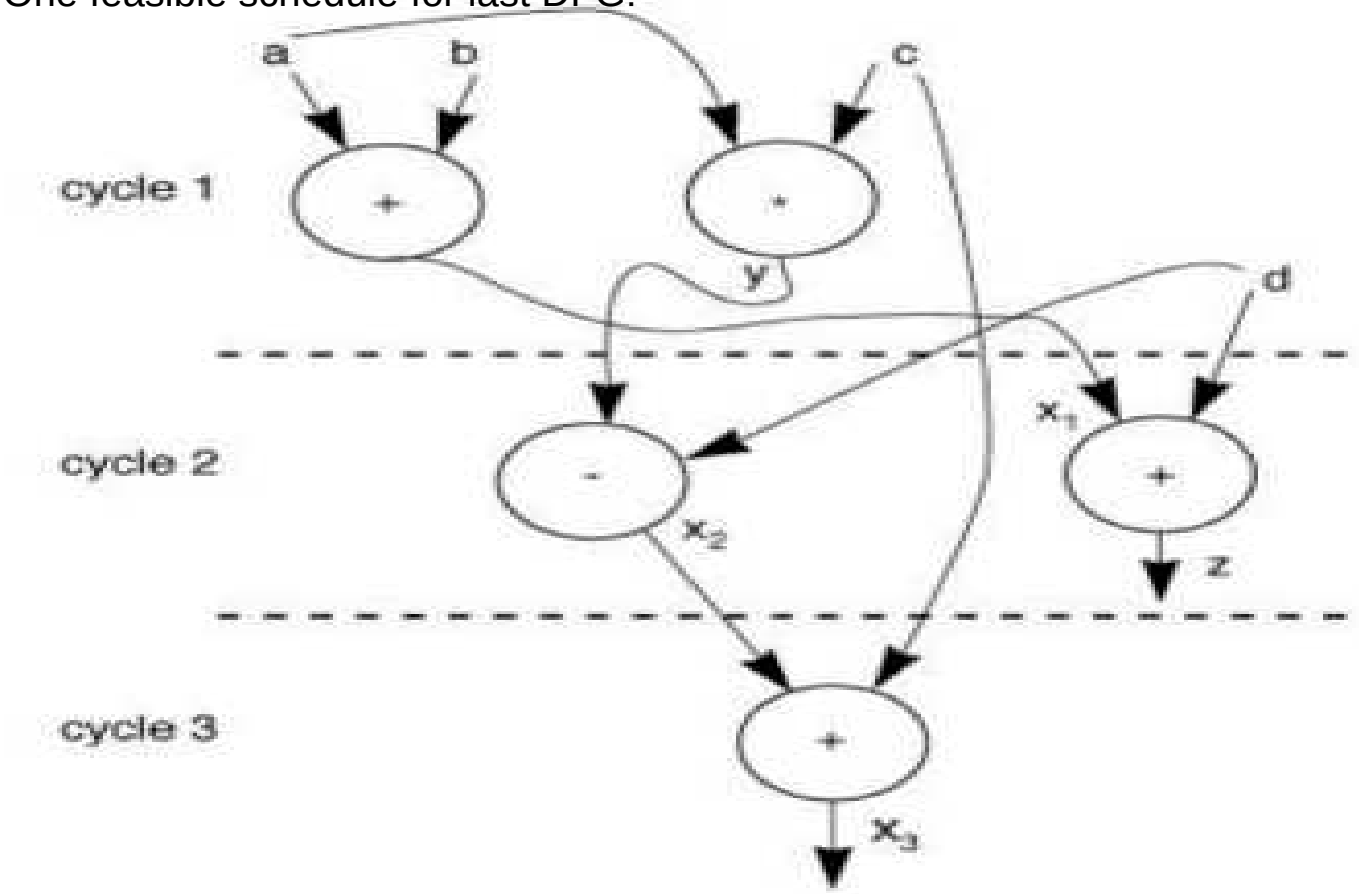


Goals of scheduling and allocation

- Preserve behavior—at end of execution, should have received all outputs, be in proper state (ignoring exact times of events)
- Utilize hardware efficiently
- Obtain acceptable performance

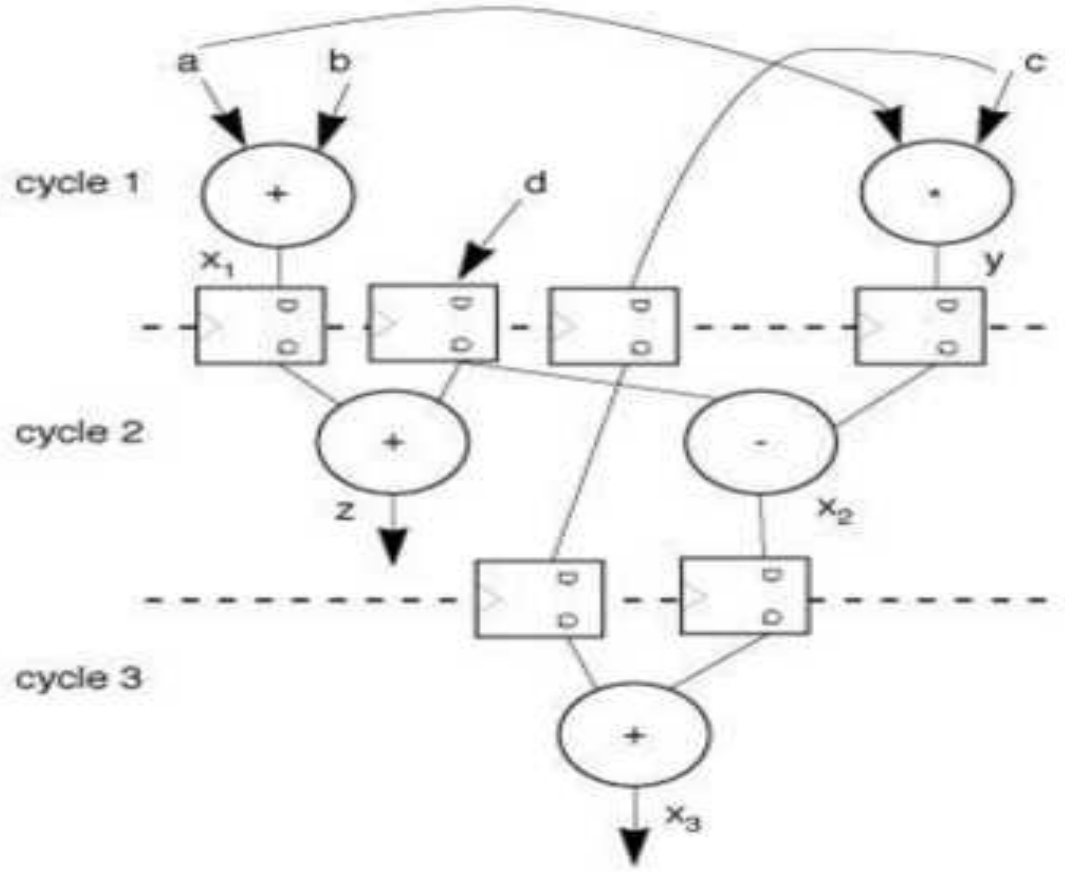
Data flow to data path-controller

One feasible schedule for last DFG:



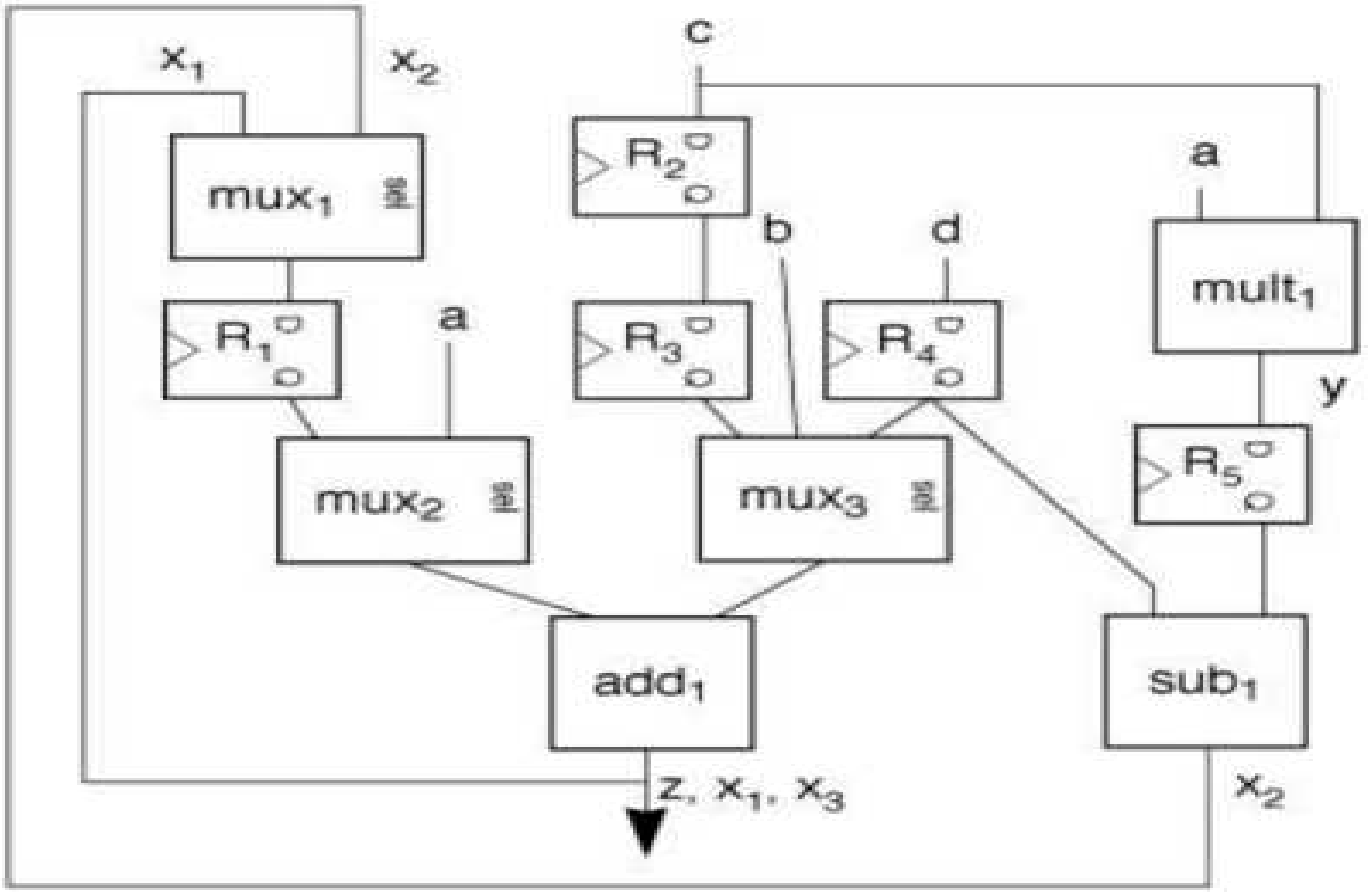
Binding values to registers

registers fall on clock cycle boundaries

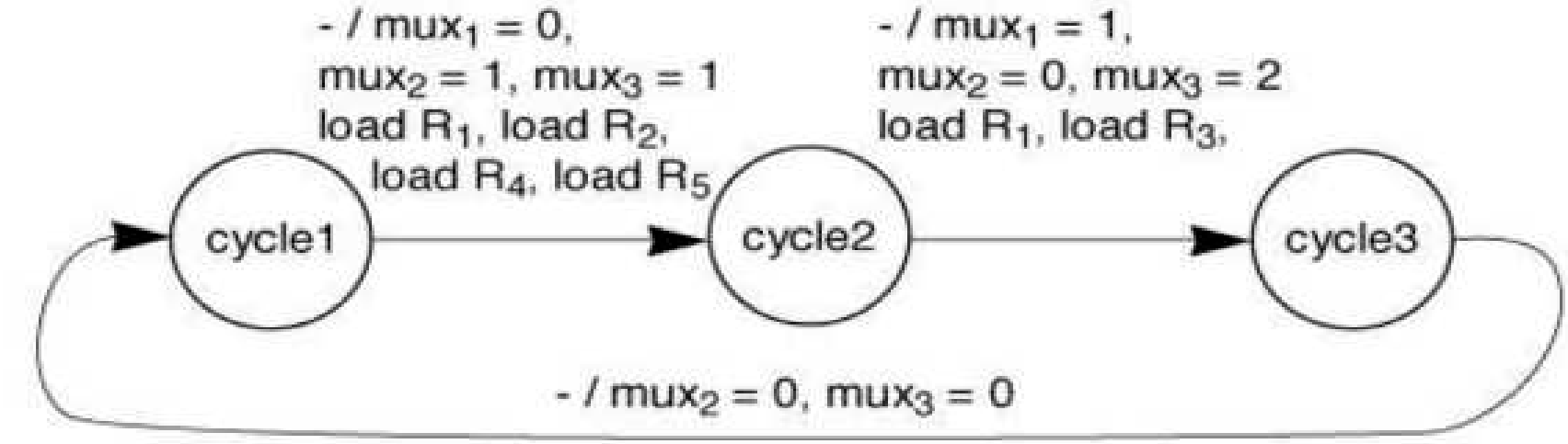


Choosing function units

muxes allow function units to be shared for several operations



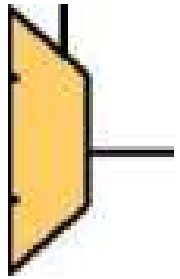
Building the sequencer



Sequencer requires three states, even with no conditionals

Behavioral Optimization

- Techniques used in software compilation
  - Expression tree height reduction
  - Constant and variable propagation
  - Common sub-expression elimination
  - Dead-code elimination
  - Operator strength reduction



- Typical Hardware transformations
  - Conditional expansion
    - If (c) then x=A else x=B c
    - compute A and B in parallel, x=(C)?A:B A .
  - Loop expansion B .
    - Instead of three iterations of a loop, replicate the loop body three times

Source: R. Gupta

Things to remember

- Design costs (hardware & software) dominate
- Within these costs verification and validation costs dominate
- IP reuse is essential to prevent design-team sizes from exploding

design cost = number of engineers x time to design

New mind set:

Design affects everything!

- A good design methodology
  - Can keep up with changing specs
  - Permits architectural exploration
  - Facilitates verification and debugging
  - Eases changes for timing closure
  - Eases changes for physical design
  - Promotes reuse

^ It is essential to

Design for Correctness

Source: Arvind, MIT