EE 5301 – VLSI Design Automation I

Part VII: High Level Synthesis

Kia Bazargan

University of Minnesota

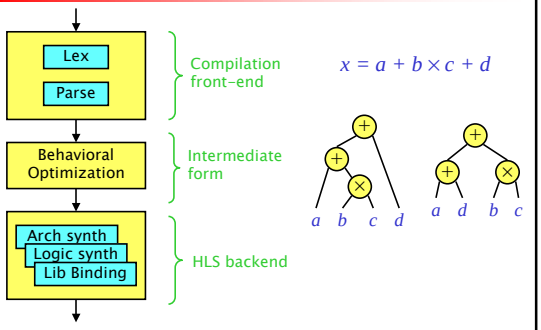
References and Copyright

- Textbooks referred (none required)
  - [Mic94] G. De Micheli
    "Synthesis and Optimization of Digital Circuits"
    McGraw-Hill, 1994.
- Slides used: (*Modified by Kia when necessary*)
  - [©Gupta] © Rajesh Gupta
    UC-Irvine
    http://www.ics.uci.edu/~rgupta/ics280.html

High Level Synthesis (HLS)

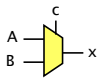
- The process of converting a high-level description of a design to a netlist
  - Input:
    - High-level languages (e.g., C)
    - Behavioral hardware description languages (e.g., VHDL)
    - Structural HDLs (e.g., VHDL)
    - State diagrams / logic networks
  - Tools:
    - Parser
    - Library of modules
  - Constraints:
    - Area constraints (e.g., # modules of a certain type)
    - Delay constraints (e.g., set of operations should finish in λ clock cycles)
  - Output:
    - Operation scheduling (time) and binding (resource)
    - Control generation and detailed interconnections

High-Level Synthesis Compilation Flow



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 (e.g., *4 → << 2)
- Typical Hardware transformations
  - Conditional expansion
    - If (c) then x=A else x=B → compute A and B in parallel, x=(C)?A:B
  - Loop expansion
    - Instead of three iterations of a loop, replicate the loop body three times



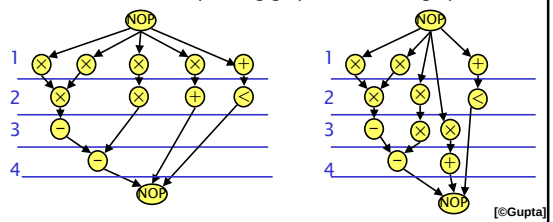
Architectural Synthesis

- Deals with "computational" behavioral descriptions
  - Behavior as sequencing graph (aka dependency graph, or data flow graph DFG)
  - Hardware resources as library elements
    - Pipelined or non-pipelined
    - Resource performance in terms of execution delay
  - Constraints on operation timing
  - Constraints on hardware resource availability
  - Storage as registers, data transfer using wires
- Objective
  - Generate a synchronous, single-phase clock circuit
  - Might have multiple feasible solutions (explore tradeoff)
  - Satisfy constraints, minimize objective:
    - Maximize performance subject to area constraint
    - Minimize area subject to performance constraints

[©Gupta]

Synthesis in Temporal Domain

- Scheduling and binding can be done in different orders or together
- Schedule:
  - Mapping of operations to time slots (cycles)
  - A scheduled sequencing graph is a labeled graph



[©Gupta]

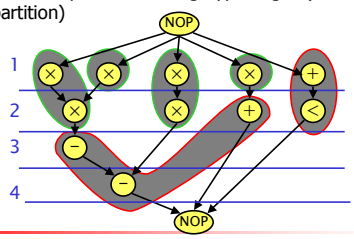
Operation Types

- For each operation, define its *type*.
- For each resource, define a resource type, and a delay (in terms of # cycles)
- T is a relation that maps an operation to a resource type that can implement it
  - $T : V \rightarrow \{1, 2, ..., n_{res}\}$.
- More general case:
  - A resource type may implement more than one operation type (e.g., ALU)
- Resource binding:
  - Map each operation to a resource with the same type
  - Might have multiple options

[©Gupta]

Schedule in Spatial Domain

- Resource sharing
  - More than one operation bound to same resource
  - Operations have to be serialized
  - Can be represented using hyperedges (define vertex partition)



[©Gupta]

Scheduling and Binding

- Resource constraints:
  - Number of resource instances of each type $\{a_k : k=1, 2, ..., n_{res}\}$.
- Scheduling:
  - Labeled vertices $\phi(v_3)=1$.
- Binding:
  - Hyperedges (or vertex partitions) $\beta(v_2)=adder1$.
- Cost:
  - Number of resources ≈ area
  - Registers, steering logic (Muxes, busses), wiring, control unit
- Delay:
  - Start time of the "sink" node
  - Might be affected by steering logic and schedule (control logic) – resource-dominated vs. ctrl-dominated

Architectural Optimization

- Optimization in view of design space flexibility
- A multi-criteria optimization problem:
  - Determine schedule ϕ and binding β.
  - Under area A, latency λ and cycle time τ objectives
- Find non-dominated points in solution space
- Solution space tradeoff curves:
  - Non-linear, discontinuous
  - Area / latency / cycle time (more?)
- Evaluate (estimate) cost functions
- Unconstrained optimization problems for resource dominated circuits:
  - Min area: solve for minimal binding
  - Min latency: solve for minimum λ scheduling

[©Gupta]

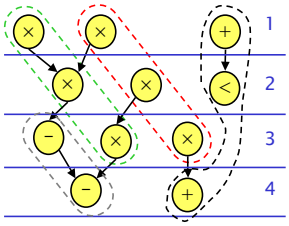
Scheduling and Binding

- Cost λ and A determined by both ϕ and β.
  - Also affected by floorplan and detailed routing
- β affected by ϕ:
  - Resources cannot be shared among concurrent ops
- ϕ affected by β:
  - Resources cannot be shared among concurrent ops
  - When register and steering logic delays added to execution delays, might violate cycle time.
- Order?
  - Apply either one (scheduling, binding) first

[©Gupta]

How Is the Datapath Implemented?

- Assuming the following schedule and binding
- Wires between modules?
- Input selection?
- How does binding / scheduling affect congestion?
- How does binding / scheduling affect steering logic?



Operation Scheduling

- Input:
  - Sequencing graph G(V, E), with n vertices
  - Cycle time τ.
  - Operation delays $D = \{d_i: i=0..n\}$.
- Output:
  - Schedule ϕ determines start time t_i of operation v_i.
  - Latency $\lambda = t_n - t_0$.
- Goal: determine area / latency tradeoff
- Classes:
  - Non-hierarchical and unconstrained
  - Latency constrained
  - Resource constrained
  - Hierarchical

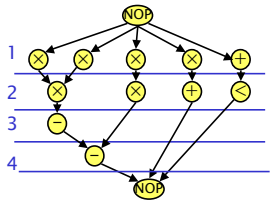
[©Gupta]

Min Latency Unconstrained Scheduling

- Simplest case: no constraints, find min latency
- Given set of vertices V, delays D and a partial order > on operations E, find an integer labeling of operations $\phi: V \rightarrow Z^+$ Such that:
  - $t_i = \phi(v_i)$.
  - $t_i \geq t_j + d_j \quad \forall (v_j, v_i) \in E$.
  - $\lambda = t_n - t_0$ is minimum.
- Solvable in polynomial time
- Bounds on latency for resource constrained problems
- ASAP algorithm used: topological order

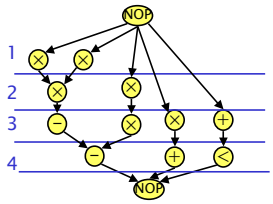
ASAP Schedules

- Schedule v_0 at $t_0=0$.
- While (v_n not scheduled)
  - Select v_i with all scheduled predecessors
  - Schedule v_i at t_i = max $\{t_j+d_j\}$, v_j being a predecessor of v_i.
- Return t_n.



ALAP Schedules

- Schedule v_n at $t_0=\lambda$.
- While (v_0 not scheduled)
  - Select v_i with all scheduled successors
  - Schedule v_i at t_i = min $\{t_j-d_j\}$, v_j being a succecessor of v_i.



Resource Constraint Scheduling

- Constrained scheduling
  - General case NP-complete
  - Minimize latency given constraints on area or the resources (ML-RCS)
  - Minimize resources subject to bound on latency (MR-LCS)
- Exact solution methods
  - ILP: Integer Linear Programming
  - Hu's heuristic algorithm for identical processors
- Heuristics
  - List scheduling
  - Force-directed scheduling

ILP Formulation of ML-RCS

- Use binary decision variables
  - $i = 0, 1, ..., n$
  - $l = 1, 2, ..., \lambda'+1$ λ' given upper-bound on latency
  - $x_{il} = 1$ if operation i starts at step l, 0 otherwise.
- Set of linear inequalities (constraints), and an objective function (min latency)
- Observations
  - $x_{il} = 0 \quad for \quad l < t_i^S \quad and \quad l > t_i^L$

    $(t_i^S = ASAP(v_i), t_i^L = ALAP(v_i))$
  - $t_i = \sum_l l.x_{il}$ t_i = start time of op i.
  - $\sum_{m=l-d_i+1}^{l} x_{im} \stackrel{?}{=} 1 \Rightarrow$ is op v_i (still) executing at step l?

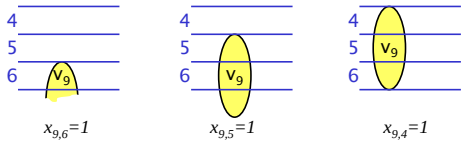
[Mic94] p.198

Start Time vs. Execution Time

- For each operation v_i, only one start time
- If $d_i=1$, then the following questions are the same:
  - Does operation v_i **start** at step l?
  - Is operation v_i **running** at step l?
- But if $d_i>1$, then the two questions should be formulated as:
  - Does operation v_i **start** at step l?
    - Does $x_{il} = 1$ hold?
  - Is operation v_i **running** at step l?
    - Does the following hold? $\sum_{m=l-d_i+1}^{l} x_{im} \stackrel{?}{=} 1$

Operation v_i Still Running at Step l ?

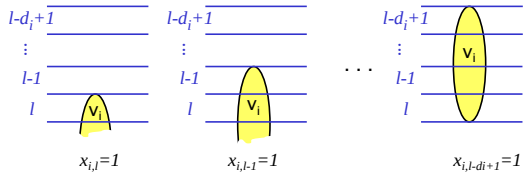
- Is v_9 running at step 6?
  - Is $x_{9,6} + x_{9,5} + x_{9,4} = 1$?



- Note:
  - Only one (if any) of the above three cases can happen
  - To meet resource constraints, we have to ask the same question for ALL steps, and ALL operations of that type

Operation v_i Still Running at Step l ?

- Is v_i running at step l ?
  - Is $x_{i,l} + x_{i,l-1} + \ldots + x_{i,l-di+1} = 1$?



ILP Formulation of ML-RCS (cont.)

- Constraints:
  - Unique start times: $\sum_l x_{il} = 1, \quad i = 0,1,\ldots,n$
  - Sequencing (dependency) relations must be satisfied

$$t_i \geq t_j + d_j \;\; \forall (v_j, v_i) \in E \Rightarrow \sum_l l.x_{il} \geq \sum_l l.x_{jl} + d_j$$

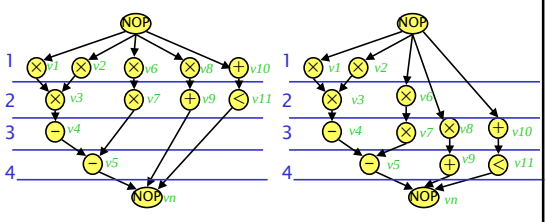
  - Resource constraints

$$\sum_{i:T(v_i)=k} \sum_{m=l-d_i+1}^{l} x_{im} \leq a_k, \quad k = 1,\ldots,n_{res}, \quad l = 1,\ldots,\bar{\lambda}+1$$

- Objective: min $\boldsymbol{c}^T\boldsymbol{t}$.
  - $\boldsymbol{t}$ =start times vector, $\boldsymbol{c}$ =cost weight (e.g., [0 0 ... 1])
  - When $\boldsymbol{c}$ =[0 0 ... 1], $\boldsymbol{c}^T\boldsymbol{t} = \sum_l l.x_{nl}$

ILP Example

- Assume $\bar{\lambda}$ = 4
- First, perform ASAP and ALAP
  - (we can write the ILP without ASAP and ALAP, but using ASAP and ALAP will simplify the inequalities)



ILP Example: Unique Start Times Constraint

- Without using ASAP and ALAP values:

$$x_{1,1} + x_{1,2} + x_{1,3} + x_{1,4} = 1$$
$$x_{2,1} + x_{2,2} + x_{2,3} + x_{2,4} = 1$$
...
...
...
$$x_{11,1} + x_{11,2} + x_{11,3} + x_{11,4} = 1$$

- Using ASAP and ALAP:

$$x_{1,1} = 1$$
$$x_{2,1} = 1$$
$$x_{3,2} = 1$$
$$x_{4,3} = 1$$
$$x_{5,4} = 1$$
$$x_{6,1} + x_{6,2} = 1$$
$$x_{7,2} + x_{7,3} = 1$$
$$x_{8,1} + x_{8,2} + x_{8,3} = 1$$
$$x_{9,2} + x_{9,3} + x_{9,4} = 1$$
....

ILP Example: Dependency Constraints

- Using ASAP and ALAP, the non-trivial inequalities are: (assuming unit delay for + and *)

$$2.x_{7,2} + 3.x_{7,3} - x_{6,1} - 2.x_{6,2} - 1 \geq 0$$
$$2.x_{9,2} + 3.x_{9,3} + 4.x_{9,4} - x_{8,1} - 2.x_{8,2} - 3.x_{8,3} - 1 \geq 0$$
$$2.x_{11,2} + 3.x_{11,3} + 4.x_{11,4} - x_{10,1} - 2.x_{10,2} - 3.x_{10,3} - 1 \geq 0$$
$$4.x_{5,4} - 2.x_{7,2} - 3.x_{7,3} - 1 \geq 0$$
$$5.x_{n,5} - 2.x_{9,2} - 3.x_{9,3} - 4.x_{9,4} - 1 \geq 0$$
$$5.x_{n,5} - 2.x_{11,2} - 3.x_{11,3} - 4.x_{11,4} - 1 \geq 0$$

ILP Example: Resource Constraints

- Resource constraints (assuming 2 adders and 2 multipliers)

$$x_{1,1} + x_{2,1} + x_{6,1} + x_{8,1} \leq 2$$
$$x_{3,2} + x_{6,2} + x_{7,2} + x_{8,2} \leq 2$$
$$x_{7,3} + x_{8,3} \leq 2$$
$$x_{10,1} \leq 2$$
$$x_{9,2} + x_{10,2} + x_{11,2} \leq 2$$
$$x_{4,3} + x_{9,3} + x_{10,3} + x_{11,3} \leq 2$$
$$x_{5,4} + x_{9,4} + x_{11,4} \leq 2$$

- Objective:
  - Since λ=4 and sink has no mobility, any feasible solution is optimum, but we can use the following anyway:

$$Min \quad x_{n,1} + 2.x_{n,2} + 3.x_{n,3} + 4.x_{n,4}$$

ILP Formulation of MR-LCS

- Dual problem to ML-RCS
- Objective:
  - Goal is to optimize total resource usage, **a**.
  - Objective function is $c^T a$, where entries in c are respective area costs of resources
- Constraints:
  - Same as ML-RCS constraints, plus:
  - Latency constraint added:

$$\sum_l l.x_{nl} \le \overline{\lambda} + 1$$

  - Note: unknown a_k appears in constraints.

[©Gupta]

Hu's Algorithm

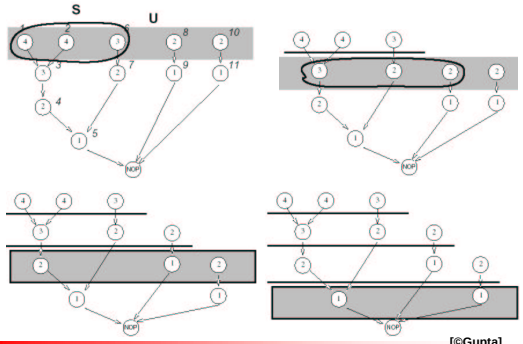
- Simple case of the scheduling problem
  - Operations of unit delay
  - Operations (and resources) of the same type
- Hu's algorithm
  - Greedy
  - Polynomial AND optimal
  - Computes lower bound on number of resources for a given latency
    OR: computes lower bound on latency subject to resource constraints
- Basic idea:
  - Label operations based on their distances from the sink
  - Try to schedule nodes with higher labels first (i.e., most "critical" operations have priority)

[©Gupta]

Hu's Algorithm

```
HU (G(V,E), a) {
    Label the vertices // label = length of longest path
                          passing through the vertex
    l = 1
    repeat {
        U = unscheduled vertices in V whose
            predecessors have been scheduled
              (or have no predecessors)
        Select S ⊆ U such that |S| ≤ a and labels in S
            are maximal
        Schedule the S operations at step l by setting
            t_i=l, i: v_i ∈ S.
        l = l + 1
    } until v_n is scheduled.
}
```

Hu's Algorithm: Example



[©Gupta]

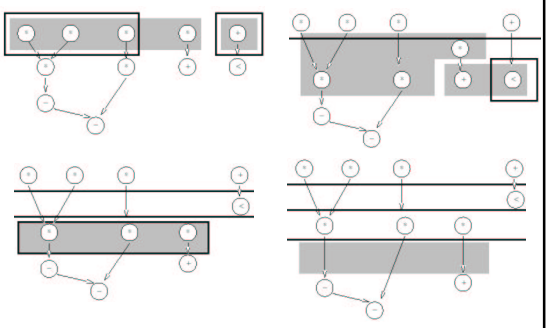
List Scheduling

- Greedy algorithm for ML-RCS and MR-LCS
  - Does NOT guarantee optimum solution
- Similar to Hu's algorithm
  - Operation selection decided by criticality
  - O(n) time complexity
- More general input
  - Resource constraints on different resource types

List Scheduling Algorithm: ML-RCS

```
LIST_L (G(V,E), a) {
    l = 1
    repeat {
        for each resource type k {
            U_{l,k} = available vertices in V.
            T_{l,k} = operations in progress.
            Select S_k ⊆ U_{l,k} such that |S_k| + |T_{l,k}| ≤ a_k
            Schedule the S_k operations at step l
        }
        l = l + 1
    } until v_n is scheduled.
}
```

List Scheduling Example



[©Gupta]

List Scheduling Algorithm: MR-LCS

LIST_R (G(V,E), λ') {
  $\boldsymbol{a} = \mathbf{1}$, $l = 1$
  Compute the ALAP times $\boldsymbol{t}^L$.
  if $t_0^L < 0$
    return (not feasible)
  repeat {
    for each resource type k {
      $U_{l,k}$ = available vertices in V.
      Compute the slacks { $s_i = t_i^L - l$, $\forall v_i \in U_{l,k}$ }.
      Schedule operations with zero slack, update $\boldsymbol{a}$
      Schedule additional $S_k \subseteq U_{l,k}$ under $\boldsymbol{a}$ constraints
    }
    $l = l + 1$
  } until v_n is scheduled.
}

Force-Directed Scheduling

- Similar to list scheduling
  - Can handle ML-RCS and MR-LCS
  - For ML-RCS, schedules step-by-step
  - BUT, selection of the operations tries to find the *globally* best set of operations
- Idea:
  - Find the mobility $\mu_i = t_i^L - t_i^S$ of operations
  - Look at the operation type probability distributions
  - Try to flatten the operation type distributions
- Definition: operation probability density
  - $p_i(l)$ = Pr { v_i starts at step l }.
  - Assume uniform distribution:

$$p_i(l) = \frac{1}{\mu_i + 1} \quad for\ l \in [t_i^S, t_i^L]$$

[©Gupta]

Force-Directed Scheduling: Definitions

- Operation-type distribution (NOT normalized to 1)
  - $q_k(l) = \sum_{i:T(v_i)=k} p_i(l)$
- Operation probabilities over control steps:
  - $p_i = \{p_i(0), p_i(1), \ldots, p_i(n)\}$
- Distribution graph of type *k* over all steps:
  - $\{q_k(0), q_k(1), \ldots, q_k(n)\}$
  - $q_k(l)$ can be thought of as *expected* operator cost for implementing operations of type *k* at step *l*.

Example

$q_{add}(1) = \frac{1}{3} = 0.33$

$q_{add}(2) = \frac{1}{3} + \frac{1}{3} + \frac{1}{3} = 1$

$q_{add}(3) = 1 + \frac{1}{3} + \frac{1}{3} + \frac{1}{3} = 2$

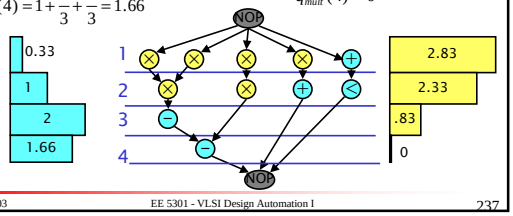
$q_{add}(4) = 1 + \frac{1}{3} + \frac{1}{3} = 1.66$

$q_{mult}(1) = 1 + 1 + \frac{1}{2} + \frac{1}{3} = 2.83$

$q_{mult}(2) = 1 + \frac{1}{2} + \frac{1}{2} + \frac{1}{3} = 2.33$

$q_{mult}(3) = \frac{1}{2} + \frac{1}{3} = 0.83$

$q_{mult}(4) = 0$



Force-Directed Scheduling Algorithm: Idea

- Very similar to LIST_L(G(V,E), **a**)
  - Compute mobility of operations using ASAP and ALAP
  - Computer operation probabilities and type distributions
  - Select and schedule operations
  - Update operation probabilities and type distributions
  - Go to next control step
- Difference with list sched in selecting operations
  - Select operations with least force
  - Consider the effect on the type distribution
  - Consider the effect on successor nodes and their type distributions

To Probe Further...

- Linear programming
  - http://www.cs.sunysb.edu/~algorith/files/linear-programming.shtml
- Linear programming tools
  - http://www-unix.mcs.anl.gov/otc/Guide/faq/linear-programming-faq.html
- Automatic complication of pipelined designs
  - T. Maruyama and T. Hoshino, "A C to HDL Compiler for Pipeline Processing on FPGAs", IEEE Symposium on FPGAs for Custom Computing Machines (FCCM), pp. 101-110, 2001.