



Local timing transformations

- **Timing modifications**
 - **global modifications - neighbors are affected significantly**
 - **local modifications - the timing of the interfaces changes very little if at all**

- **Local timing transformations**
 - **how effectively are used resources of a module**
 - **pipelining**
 - **retiming**

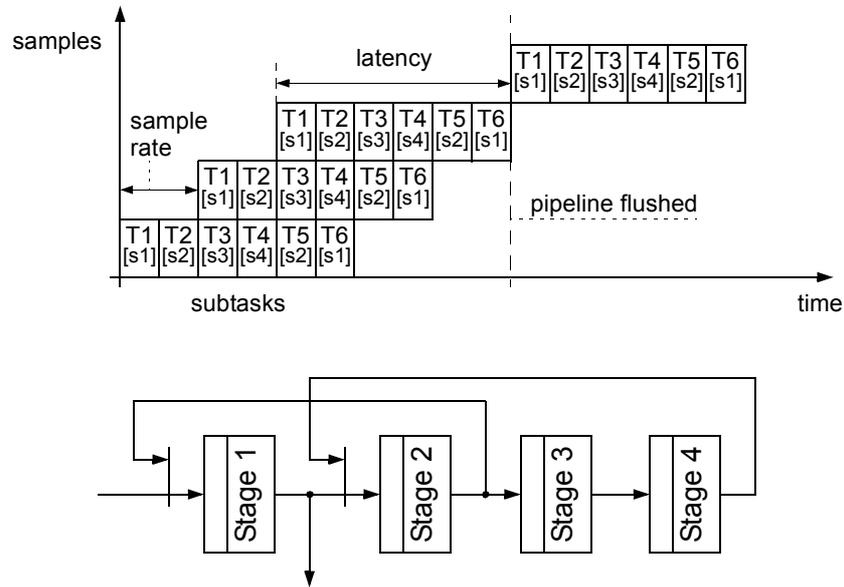


Pipelining

- ***Pipelining* - an implementation technique whereby multiple instructions are overlapped in execution**
- ***Latency (L)* - total number of time units needed to complete the computation on one input sample**
- ***Sample rate (R)* - the number of time units between two consecutive initiations, where initiation is the start of a computation on an input sample**
- ***A (pipe) stage* is a piece of HW that is capable of executing certain subtask of the computation**
- ***The reservation table* is a two-dimensional representation of the data flow during one computation. One dimension corresponds to the stages, and the other dimension corresponds to time units.**

- **Actions in pipeline: *flushing, refilling, stalling.***

Pipeline - example



Pipeline measurements

- **Average initiation rate (measure of pipeline performance):**

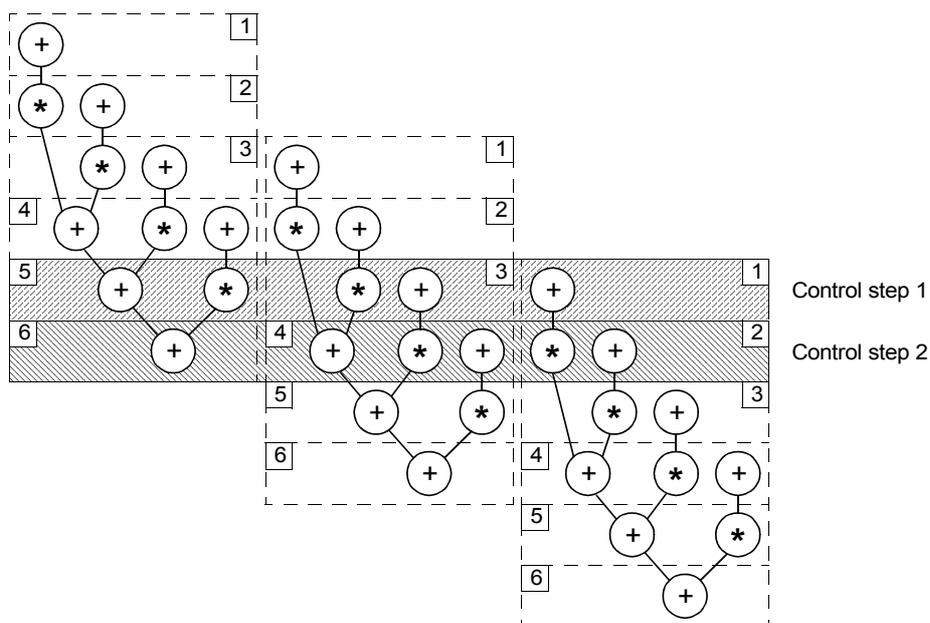
$$R_{\text{init}, N \rightarrow \infty} = 1 / (R \times t_{\text{stage}} + r_{\text{synchro}} \times (L-R) \times t_{\text{stage}})$$

- **R** - sample rate, **L** - latency,
- **t_{stage}** - the time one stage needs to complete its subtask,
- **r_{synchro}** = **N_{flush}** / **N** - resynchronization rate,
- **N_{flush}** - the number of input samples that cause flushing,
- **N** - number of input samples.

Functional pipelining

- In conventional pipelining, stages have physical equivalents, i.e. the stage hardware is either shared completely in different time units or not shared at all.
- In the case of large functional units, there is no physical stage corresponding to the logical grouping of operations in a time step.
- A *control step* corresponds to a group of time steps that overlap in time. Operations belonging to different control steps may share functional units without conflict.
- Operations, belonging to the time steps $s+n \times L$, for $n \geq 0$, are executed simultaneously and cannot share hardware.

Functional pipelining of 8-point FIR filter





Pipelining - conclusion

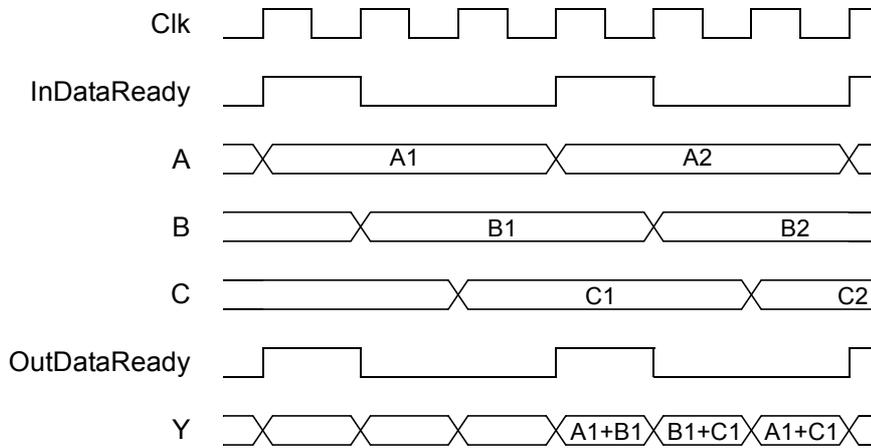
- As in conventional scheduling, the decision problem for pipeline scheduling is NP-complete.
- Solved heuristically (modified ILP for RCS or modified force-directed scheduling for TCS).
- In case of pipelined scheduling, at least $|R_k| = \lceil |\{o \in V: o \in r_k\}| / L \rceil$
- FUs of type k have to be provided to ensure that a schedule for the given latency L exists.



Pipeline: example

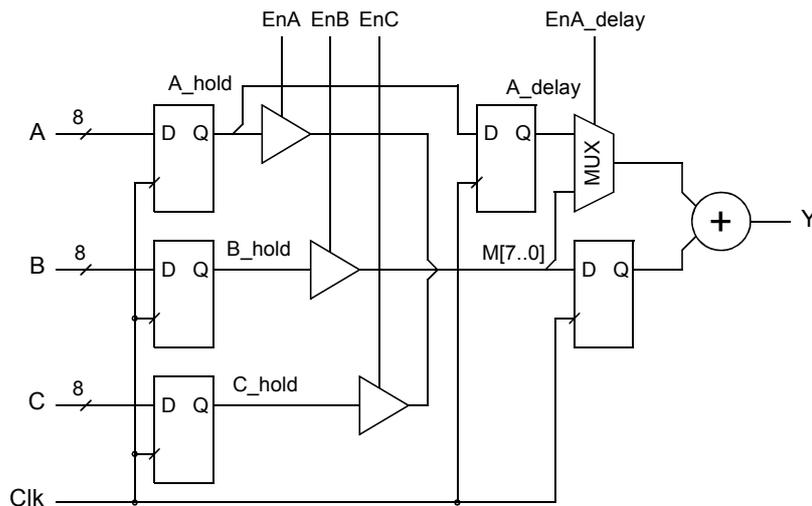
- **Design task**
 - There are 8-bit busses - A, B and C - that have valid data arriving in three consecutive clock cycles.
 - Design a model that computes the sum of the three pairs of busses, i.e., (A+B), (A+C), and (B+C); and supplies the results onto a single 9-bit output bus in three consecutive clock cycles.
 - On a separate input signal (InDataReady), logic value 1 for one clock cycle is used to indicate when there is a valid data on the first bus (A). On the following two consecutive clock cycles, data is valid on the other busses - B and C.
 - On a separate output signal (OutDataReady), there should be logic value 1 for one clock cycle to indicate the first of the three clock cycles when the three summed output results are available.
- **Constraint**
 - Chip area is critical

Pipeline: expected timing



- **Subtask**
 - Describe the data path at RTL level

Pipeline: data path



- **Subtask**
 - Describe control and data paths in VHDL as separate modules



Pipeline: control path (VHDL)

```

process (clk, reset) begin
  if (Reset = '1') then
    InDataReady_delay <= '0';
    EnA <= '1'; -- then there is active driver
    EnA_delay <= '0';
    EnB <= '0'; EnC <= '0';
  elsif rising_edge(clk) then
    InDataReady_delay <= InDataReady;
    EnA <= InDataReady or not (inDataReady_delay or EnB);
    EnB <= InDataReady_delay;
    EnC <= EnB; EnA_delay <= EnC;
  end if;
end process;
OutDataReady <= EnC;

```



Pipeline: data path (VHDL)

```

process(clk) begin
  if rising_edge(clk) then
    A_hold <= A; B_hold <= B; C_hold <= C;
    A_delay <= A_hold;
    M_delay <= M;
    if (EnA_delay = '1') then
      Y <= ('0' & M_delay) + A_delay;
    else Y <= ('0' & M_delay) + M;
    end if;
  end if;
end process;
M <= A_hold when EnA = '1' else (others => 'Z');
M <= B_hold when EnB = '1' else (others => 'Z');
M <= C_hold when EnC = '1' else (others => 'Z');

```



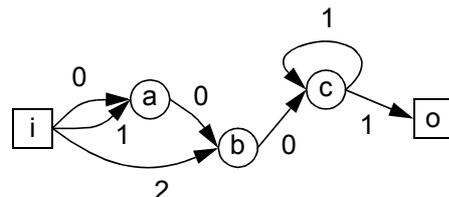
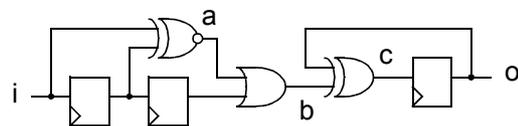
Retiming

- **Minimization of the cycle-time or the area of synchronous circuits by changing the position of the registers**
 - cycle-time \leftarrow critical path
- **The number of registers may increase or decrease**
 - area minimization corresponds to minimizing the number of registers
 - combinational circuits are not affected (almost)
- **Synchronous logic network**
 - variables
 - boolean equations
 - synchronous delay annotation



Synchronous logic network

- **State-based model**
 - transition diagrams or tables
 - explicit notion of state
 - implicit notion of area and delay
- **Structural model**
 - synchronous logic network
 - implicit notion of state
 - explicit notion of area and delay





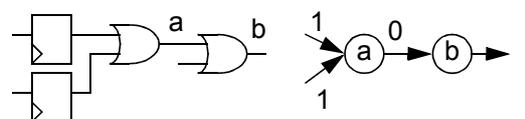
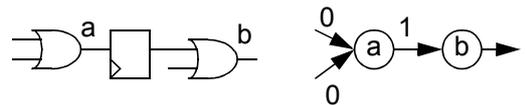
Optimizations

- **Optimize combinational logic only**
 - logic minimization
 - two-level minimization
 - multi-level minimization
- **Optimize register position only**
 - retiming
- **Optimize overall circuit**
 - peripheral retiming
 - synchronous transformations
 - algebraic
 - boolean



Optimization steps

- **Separate registers from combination logic**
- **Optimize combinational logic**
 - modify equations
 - modify graph structure
- **Connect registers back to the network**
- **Move register positions**
 - do not modify combinational logic
- **Preserve network structure**
 - modify weights
 - do not modify graph structure





Assumptions

- **Node delay is constant**
 - no fanout delay dependency
- **Graph topology is invariant**
 - no logic transformations
- **Synchronous implementation**
 - cycles have positive weights
 - edges have non-negative weights
- **Consider topological paths**
 - no false path analysis

- **Retiming of a node**
 - integer, register moved from output to input
- **Retiming of a network**
 - vector of node retiming
- **A family of equivalent networks = the original network + retiming vectors**



Definitions & properties

- $w(v_i, v_j)$ - weight of an edge (v_i, v_j)
- (v_i, \dots, v_j) - path from v_i to v_j
- $d(v_i, \dots, v_j)$ - path delay from v_i to v_j

- **retiming of an edge:** $\bar{w}_{ij} = w_{ij} + r_j - r_i$
- **retiming of a path:** $\bar{w}(v_i, \dots, v_j) = w(v_i, \dots, v_j) + r_j - r_i$
- **cycle weights are invariant**

Legal retiming

- **clock period ϕ**
- **no edge weight is negative:** $\bar{w}_{ij} = w_{ij} + r_j - r_i \geq 0 \quad \forall i, j$
- **no retiming of I/O pins:** $r_i = 0 \quad \forall i \in I/O$
- **each path (v_i, \dots, v_j) with $d(v_i, \dots, v_j) \geq \phi$ has at least one register:**
 $\bar{w}(v_i, \dots, v_j) = w(v_i, \dots, v_j) + r_j - r_i \geq 1 \quad \forall i, j$



Relaxation based algorithm

- Looks for path with excessive delay
- Make them shorter by pulling closer the terminal register
 - some other paths may become too long
 - those paths whose tail has been moved
- Use an iterative approach

- Define node's data ready time
 - total delay from register boundary
- Iterative approach
 - find nodes with data ready time $> \phi$
 - retime these nodes by 1
- Properties
 - finds legal timing in at most $|N|$ iterations, if one exist



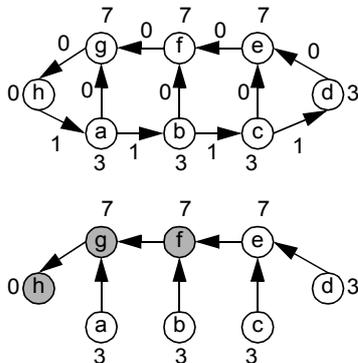
Retiming: example

$\phi = 13$

1) data ready times:

$t_a=3, t_b=3, t_c=3, t_d=3, t_e=10,$
 $t_f=17, t_g=24, t_h=24$

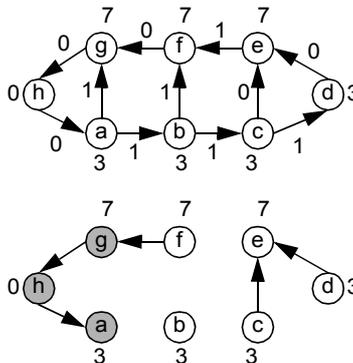
retime $\{f, g, h\}$ by 1



2) data ready times:

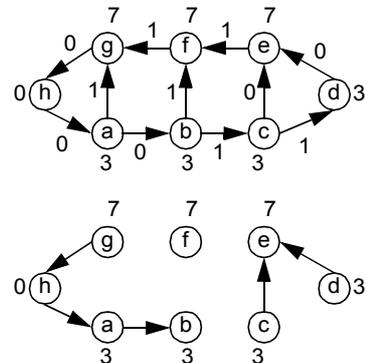
$t_a=17, t_b=3, t_c=3, t_d=3, t_e=10,$
 $t_f=7, t_g=14, t_h=14$

retime $\{a, t_g, t_h\}$ by 1



3) data ready times:

$t_a=10, t_b=13, t_c=3, t_d=3,$
 $t_e=10, t_f=7, t_g=7, t_h=7$





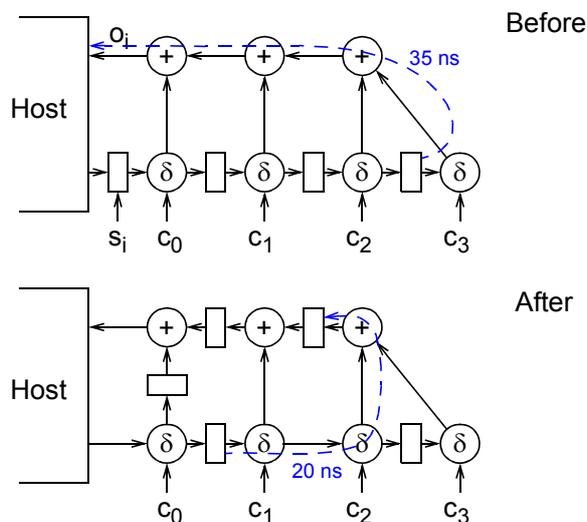
Bellman-Ford algorithm

- The shortest path problem on a weighted graph complexity - $O(|V||E|) \leq O(n^3)$
- Compute $W(u,v)$ and $D(u,v)$ for all paths in G from u to v .
- $W(u,v)$ is the minimal number of registers on any path from vertex u to vertex v in G .
- $D(u,v)$ is the maximal propagation delay on any path from vertex u to vertex v with $W(u,v)$ registers.
Sort the elements in the range of $D(u,v)$ in ascending order.
- Binary search has to be performed among the elements $D(u,v)$ for the minimal clock period.
- Complexity: $O(|V|^3 \log|V|)$



Retiming: example #2

Digital correlator



$$o_i = \sum_{j=0}^3 \delta(s_{i-j} c_j)$$



Retiming at higher abstraction levels?

- The same, in principle, as for logic networks
 - operation nodes - functions
 - delay nodes - e.g. shared resources (memories)
- More possibilities to manipulate the functions - higher complexity of the optimization task
 - partitioning/merging functions
 - reorganizing shared resources



Retiming at higher abstraction levels

- Optimization:
 - control-step #1: $r_1 \leftarrow c_1(v_i, \dots)$,
 - control-step #2: $r_2 \leftarrow c_2(r_1, v_j, \dots)$
 - r_1, r_2 - registers; c_1, c_2 - combinational blocks; v_i, v_j - variables
 - $f_{\max} = 1 / \max(\text{delay}(c_1), \text{delay}(c_2))$,
 - $\text{delay}(c_1) > \text{delay}(c_2) : \text{ then } c_1^{\text{new}} = g(f_1(v_i, \dots), f_2(v_i, \dots))$
- After resynthesis, $\text{delay}(g) + \text{delay}(c_2) < \text{delay}(c_1) :$
 - control-step #1: $r_1 \leftarrow f_1(v_i, \dots); r_x \leftarrow f_2(v_i, \dots)$
 - control-step #2: $r_2 \leftarrow c_2(g(r_1, r_x), v_j, \dots)$

