High Throughput

- High-throughput design is concerned:
  - More about the steady-state data rate
  - Less about latency (i.e., the time an specific piece of data requires to propagate through the design)
- High-throughput design employs the so-called pipeline architecture
  - New data can begin processing before the prior data has finished → improving system throughput
- Let us consider some examples to illustrate the beauty of pipeline architecture

Low latency

- Low-latency design: passes data from the input to the output as quickly as possible
  - To achieve the goal, minimizing intermediate processing delays
- Often, a low-latency design requires:
  - Parallelisms
  - Removal of pipelining
  - Logical short cuts
- Observing the previous pipelined implementation, we can see the possibility of reducing latency:
  - At each pipeline stage, the product of each multiply must wait for the next clock edge to propagate to the next stage
  - Therefore, we expect that removing pipeline registers → reducing latency

Timing

- Timing refers to the clock speed of a design
- Maximum delay between any two sequential elements in a design will determine the max clock speed.
- Maximum speed, or maximum frequency, is defined as follows:

$$F_{\text{max}} = \frac{1}{T_{\text{clk-q}} + T_{\text{logic}} + T_{\text{routing}} + T_{\text{setup}} - T_{\text{skew}}}$$

- where
  - F_{max} : maximum allowable frequency for clock
  - $T_{\text{clk-q}}$: time from clock arrival until data arrives at Q
  - T_{logic} : propagation delay through logic between flipflops
  - T_{routing} : routing delay between flipflops
  - T_{setup} : setup time
  - T_{skew} : propagation delay of clock

Timing – Add Register Layers

- **Adding intermediate layers of registers to the critical path is the first strategy for architectural timing improvement**
- This technique should be used in highly pipelined designs, where:
  - An additional clock cycle latency does not violate the design specifications
  - The overall functionality will not be affected by the further addition of registers

- For example: assume the architecture for the following FIR implementation does not meet timing:

Timing – Parallel Structures

- **Reorganizing the critical path such that logic structures are implemented in parallel is the second strategy for architectural timing improvement**
- This technique should be used whenever:
  - A function that currently evaluates through a serial string of logic
  - But can be broken and evaluated in parallel

- For instance, assume that the standard pipelined power-of-3 design discussed above does not meet timing
- To create parallel structures:
  - Break up the multipliers into independent operations
  - Recombine them to get results

Timing – Flatten Logic Structures

- **Flattening logic structures is the third strategy for architectural timing improvement**
- This technique is closely related to the idea of parallel structures, but applies specifically to logic that is chained due to priority encoding.

- Normally, synthesis and layout tools are not smart enough to break up logic structures coded in a serial fashion
- They also do not have enough information relating to the priority requirements of the design
- Therefore, it is possible that they are unable to give us a configuration with our desired timing