Application Specific Integrated Circuits for Digital Signal Processing
Lecture 6

Oscar Gustafsson

Today's topics

- Look-ahead pipelining and block processing of recursive algorithms
- Introduction to scheduling (mapping to architecture)

Decreasing the iteration period bound of recursive algorithms

- Minimal sample period (iteration period bound)

$$T_{\text{min}} = \max_i \frac{\sum_{\text{Op. in loop } i} T_{L,k}}{N_i} = \frac{1}{f_{\text{max}}} \tag{1}$$

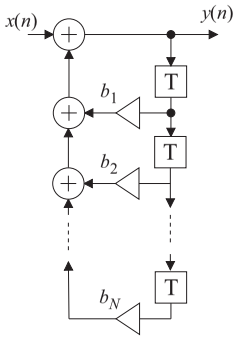
- Possible to move operations out of the loop
- Not possible to introduce pipelining in loops
- Possible to find algorithms with similar transfer function but more delay elements in the loops?

Clustered Look-Ahead Pipelining

- Consider only the recursive parts

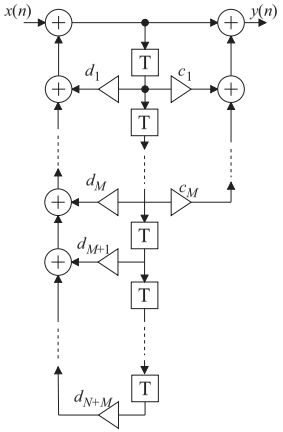
$$H(z) = \frac{1}{1 - \sum_{i=1}^{N} b_i z^{-i}} \tag{2}$$

- Direct form realization



Clustered Look-Ahead Pipelining

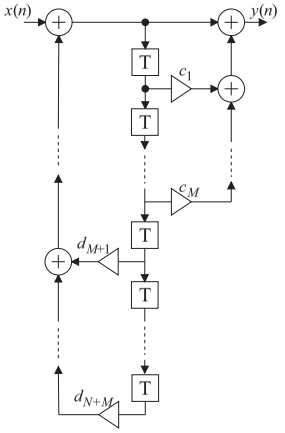
- Multiply with polynomial $1 + \sum_{i=1}^{M} c_i z^{-i}$ in both numerator and denominator



$$H(z) = \frac{1}{1 - \sum_{i=1}^{N} b_i z^{-i}} \frac{1 + \sum_{i=1}^{M} c_i z^{-i}}{1 + \sum_{i=1}^{M} c_i z^{-i}} = \frac{1 + \sum_{i=1}^{M} c_i z^{-i}}{1 - \sum_{i=1}^{M+N} d_i z^{-i}} \quad (3)$$

Clustered Look-Ahead Pipelining

- Select c_i such that $d_i = 0, i = 1, 2, \ldots M$
- A loop has at least $M + 1$ delay elements!

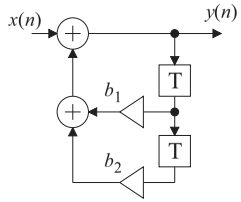


- What happens?
  - Recursive section now has M additional poles
  - FIR filter in cascade to remove the effect
  - *Pole-zero cancellation*

Clustered Look-Ahead Pipelining

- Example: Second-order section, $M = 2$

$$H(z) = \frac{1}{1 - \sum_{i=1}^{2} b_i z^{-i}} \quad (4)$$

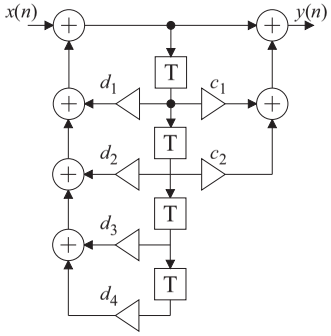


$$T_{\text{min}} = \frac{T_{L,\text{mult}} + 2T_{L,\text{add}}}{1} \quad (5)$$

Clustered Look-Ahead Pipelining

- Multiply with second order polynomial

$$H(z) = \frac{1}{1 - \sum_{i=1}^{2} b_i z^{-i}} \frac{1 + \sum_{i=1}^{2} c_i z^{-i}}{1 + \sum_{i=1}^{2} c_i z^{-i}} = \frac{1 + \sum_{i=1}^{2} c_i z^{-i}}{1 - \sum_{i=1}^{4} d_i z^{-i}} \quad (6)$$



Clustered Look-Ahead Pipelining

- Consider recursive part

$$1-\sum_{i=1}^{4} d_i z^{-i} = \left(1-\sum_{i=1}^{2} b_i z^{-i}\right)\left(1+\sum_{i=1}^{2} c_i z^{-i}\right) =$$

$$= 1-(b_1-c_1)z^{-1}-(b_2-c_2+c_1b_1)z^{-2}-(b_1c_2+c_1b_2)z^{-3}-b_2c_2z^{-4}$$

- Solve for $d_1 = d_2 = 0$:

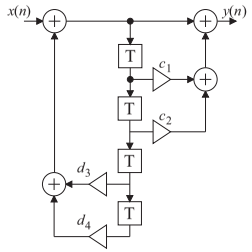
$$c_1 = b_1$$

$$c_2 = b_2 + b_1c_1 = b_2 + b_1^2$$

- Gives:

$$d_3 = b_1c_2 + c_1b_2 = b_1^3 + 2b_1b_2$$

$$d_4 = b_2c_2 = b_2^2 + b_1^2b_2$$



$$T_{\text{min}} = \frac{T_{L,\text{mult}} + 2T_{L,\text{add}}}{3}$$

Clustered Look-Ahead Pipelining

- Consider second-order section from limit cycle discussion, $b_1 = \frac{489}{256}$ and $b_2 = -\frac{15}{16}$
- We now get

$$c_1 = b_1 = \frac{489}{256} \quad (7)$$

$$c_2 = b_2 + b_1^2 = \frac{177681}{65536} \quad (8)$$

$$d_3 = b_1^3 + 2b_1b_2 = \frac{56841849}{16777216} \quad (9)$$

$$d_4 = b_2^2 + b_1^2b_2 = -\frac{2665215}{1048576} \quad (10)$$

- Poles: $-0.95508 \pm j1.34128$ and $0.95508 \pm j0.15914 \Rightarrow$ poles outside of the unit circle!
- Zeros: $-0.95508 \pm j1.34128$ (but at least perfect cancellation)

Clustered Look-Ahead Pipelining

- Quantize the coefficients to eight fractional bits

$$c_1 = \frac{489}{256} \quad (11)$$

$$c_2 = \frac{347}{128} \quad (12)$$

$$d_3 = \frac{867}{256} \quad (13)$$

$$d_4 = -\frac{651}{256} \quad (14)$$

- Poles: $-0.95511 \pm j1.34116$ and $0.95512 + j0.16057$ (changed from previous slide)
- Zeros: $-0.95508 + j1.34118$ (no exact match of the pole-zero cancellation)
- (Poles still outside the unit circle, but to illustrate the quantization effect)

Clustered Look-Ahead Pipelining

- Issues
- Finite wordlength effects
  - Rounding the coefficient values will not produce perfect cancellation
- Where did the new poles end up?
  - No control if they are within the unit circle
- Only works for direct form (although a similar concept sometimes can be applied for other structures)

Scattered Look-Ahead Pipelining

- Limit the added poles to be at the same radius as the poles under consideration
- The polynomial is formed by $M-1$ equally spaced roots
- Consider a first-order section for simplicity

$$H(z) = \frac{1}{r - z^{-1}} = \frac{1}{r - z^{-1}} \frac{\prod_{k=1}^{M-1} \left(re^{\frac{j2\pi k}{M}} - z^{-1} \right)}{\prod_{k=1}^{M-1} \left(re^{\frac{j2\pi k}{M}} - z^{-1} \right)} \tag{15}$$

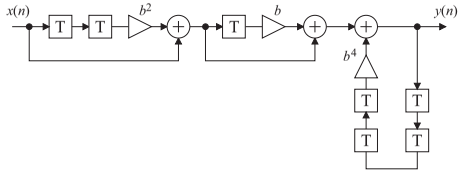
$$= \frac{\prod_{k=1}^{M-1} \left(re^{\frac{j2\pi k}{M}} - z^{-1} \right)}{r^M - z^{-M}} \tag{16}$$

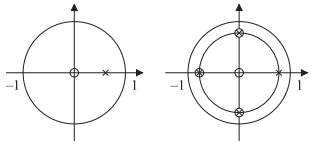
Scattered Look-Ahead Pipelining

- Example: First-order section, $M = 4$

$$H(z) = \frac{1}{b - z^{-1}} = \frac{1}{b - z^{-1}} \frac{\prod_{k=1}^{3} \left(be^{\frac{j2\pi k}{4}} - z^{-1} \right)}{\prod_{k=1}^{3} \left(be^{\frac{j2\pi k}{4}} - z^{-1} \right)}$$

$$= \frac{\left(b + z^{-1}\right)\left(1 + b^2 z^{-2}\right)}{b^M - z^{-M}}$$





Scattered Look-Ahead Pipelining

- Easier to generalize to arbitrary structures
- Denominator will be on the form z^M
- Always stable (as in poles inside the unit circle)
- Still possible issues with wordlengths

Block Processing

- Consider a state-space representation of a recursive algorithm

$$\begin{bmatrix} \mathbf{v}(n+1) \\ y(n) \end{bmatrix} = \begin{bmatrix} \mathbf{A} & \mathbf{b} \\ \mathbf{c} & d \end{bmatrix} \begin{bmatrix} \mathbf{v}(n) \\ x(n) \end{bmatrix} \tag{17}$$

- The recursive part relates to $\mathbf{A}$, i.e., updating the states from previous states
- Consider the next iteration

$$\begin{aligned} \mathbf{v}(n+2) &= \mathbf{A}\mathbf{v}(n+1) + \mathbf{b}x(n+1) && (18) \\ &= \mathbf{A}(\mathbf{A}\mathbf{v}(n) + \mathbf{b}x(n)) + \mathbf{b}x(n+1) && (19) \\ &= \mathbf{A}^2\mathbf{v}(n) + \underbrace{\mathbf{A}\mathbf{b}x(n) + \mathbf{b}x(n+1)}_{\text{non-recursive}} && (20) \end{aligned}$$

- Hence, we can update two iterations ahead using a single matrix-vector multiplication

Block Processing

- Similarly, consider the output

$$
\begin{aligned}
y(n+1) &= \mathbf{cv}(n+1) + dx(n+1) && (21)\\
&= \mathbf{c}(\mathbf{Av}(n) + \mathbf{b}x(n)) + dx(n+1) && (22)\\
&= \underbrace{\mathbf{cAv}(n) + \mathbf{cb}x(n) + dx(n+1)}_{\text{non-recursive}} && (23)
\end{aligned}
$$

Block Processing

- Generalize to M samples ahead

$$
\begin{aligned}
\mathbf{v}(n+M) &= \mathbf{Av}(n+M-1) + \mathbf{b}x(n+M-1) && (24)\\
&= \mathbf{A}(\mathbf{Av}(n+M-2) + \mathbf{b}x(n+M-2)) \\
&\quad + \mathbf{b}x(n+M-1) && (25)\\
&\dots \\
&= \mathbf{A}^M\mathbf{v}(n) + \underbrace{\sum_{k=1}^{M} \mathbf{A}^{M-k}\mathbf{b}x(n+k-1)}_{\text{non-recursive}} && (26)
\end{aligned}
$$

- Can iterate an arbitrary number of samples ahead in the same time as one iteration
- All non-recursive parts can be pipelined

Block Processing

- Issues
- Finite wordlength effects
  - In practice pole-zero cancellation is performed
  - Time-variant impulse response
    - Different inputs have different impulse responses because of coefficient quantization
- Computational complexity
  - High cost for computing the non-recursive parts

Direct synthesis

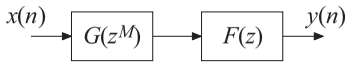
- Constrain transfer function to be on the form

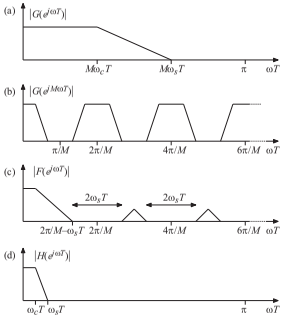
$$H(z) = \frac{B(z)}{A(z^M)} \qquad (27)$$

- Rough design path
  - Take arbitrary structure and add enough delay elements
  - Cascade with FIR filter (to cancel/compensate for the additional poles)
  - Run numerical optimization
- Practical usage varies with filter structure, required speed increase etc
  - Optimization problem is highly non-linear $\Rightarrow$ hard to find global optimum
  - Even harder to find fixed-point coefficients so quantizing the obtained coefficients may break the specification
  - The complexity benefit of IIR filters may be lost
  - Need to check that poles stay inside the unit circle, complexity of that depends on filter structure

Direct synthesis

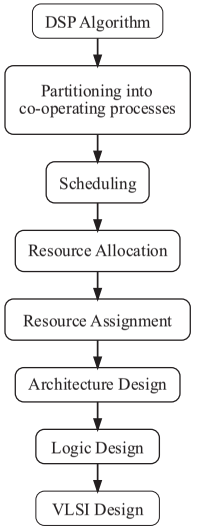
- For narrow-band and wide-band filters one can design the filters separately





- If $G(z)$ is a recursive filter, it will have an M times lower T_{min}
- For wide-band filters, compute the complementary output
- (For FIR filters this can also be used to reduce the complexity as $G(z)$ will have about M times fewer multiplications compared to $H(z)$)

Mapping algorithms to architecture



Scheduling

- When to execute a process?
- Process: arbitrary large computational task
  - Preemptive process: Can be stopped and started again at a different (or the same) processing element (PE), can be split into smaller processes
  - Non-preemptive process: opposite
- Processes are characterized by
  - Start-time and duration (deadline)
  - Random or periodic arrival
- Deadline must be met: hard real-time processes

Scheduling

- If processes are known in advance a static schedule can be derived
- If not a dynamic schedule must be used
- Dynamic schedules may cause large overhead as many scheduling problems are NP-hard
- Static scheduling can be used for most DSP algorithms
- Possible to synthesize optimal architectures from a given static schedule

Scheduling

- Processes will be mapped onto a hardware structure (processing element, PE)
- Divide in three separate stages:
  - Scheduling – place the processes in time
  - Resource allocation – determine the number of required PEs, memories, communication channels etc
  - Resource assignment – assign a process to a certain PE

Optimal scheduling

- Sample period optimal

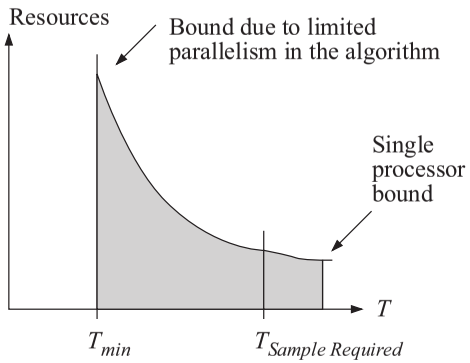
$$T_{\text{sample}} = T_{\text{min}}$$

  - No shimming delays in critical loop
- Delay optimal
  - Input to output delay (latency) = minimal input to output delay
  - No shimming delays from input to output
- PE optimal

$$N_{\text{PE}} = \left\lceil \frac{\sum T_{\text{exe}}}{T_{\text{schedule}}} \right\rceil$$

  - Compare with average number of PEs required

Resources vs T_{sample}



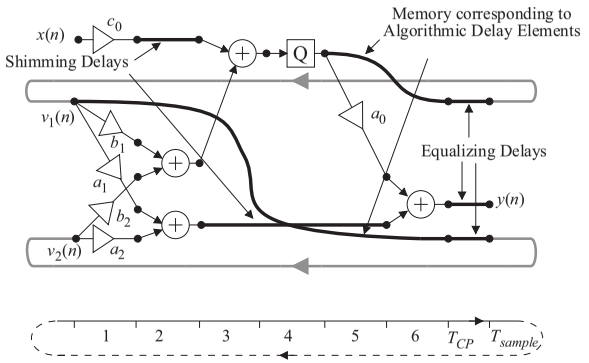
- Algorithm parallelism is only bounded for recursive algorithms

Resources vs T_{sample}

- Goal: find minimum-cost hardware structure that meets performance constraints
- Cost function?
  - Commonly: number of PEs/complexity of PEs
  - But memory is expensive and may be limiting in many applications
  - Power
- Idea: minimize the amount of hardware resources (PEs, memory, communication, and control) with the assumption that this leads to small area and low power consumption

Single Interval Scheduling

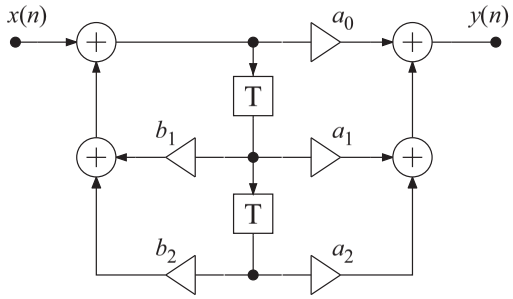
- Computation graph



- Possible to move operations as long as the ordering is not changed

Scheduling Example

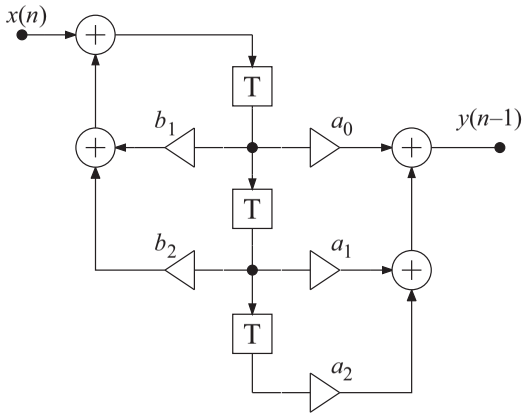
- $T_{L,\text{mult}} = T_{\text{exe,mult}} = 4$ time units (t.u.)
- $T_{L,\text{add}} = T_{\text{exe,add}} = 1$ t.u.
- Required sample period $T_{\text{sample}} = 10$ t.u.



- Minimal sample period: $T_{\text{min}} = \max\{\frac{4+1+1}{1}, \frac{4+1+1}{2}\} = 6$ t.u. **OK!**
- Critical path: $T_{\text{CP}} = 4+1+1+4+1 = 11$ t.u. **Not OK!**

Scheduling Example

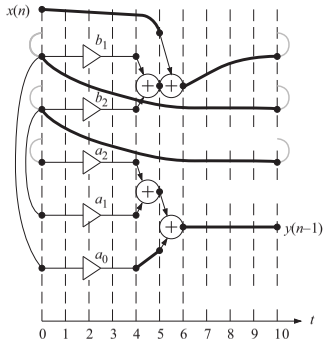
- Pipeline!



- Critical path: $T_{\text{CP}} = 4+1+1 = 6$ t.u. **OK!**

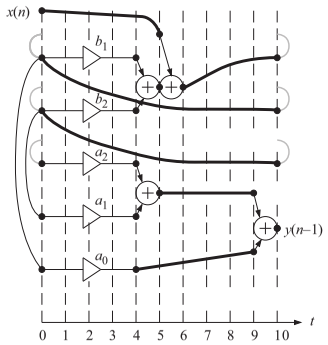
Scheduling Example

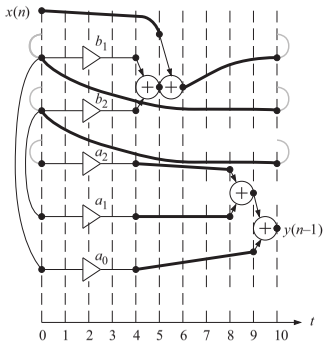
- Initial schedule obtained after determining the SFG in precedence graph and introducing timing for the operations



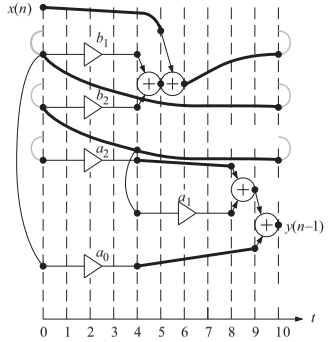
- OK to use, but require five multipliers and two adders
- Lots of idle time towards the end of the scheduling period
- Move some operations there

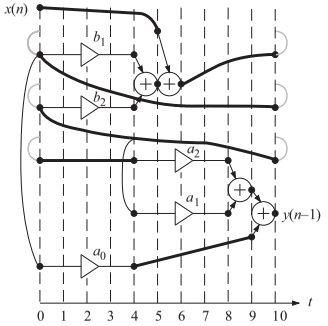
Scheduling Example





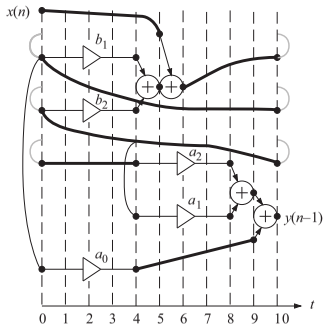
Scheduling Example





- Now three multipliers and one adder are required

Scheduling Example



- Now three multipliers and one adder are required
- Minimum number of operators?
- Multipliers: $\left\lceil \frac{\sum T_{\text{exe,mult}}}{T_{\text{schedule}}} \right\rceil = \lceil \frac{5 \times 4}{10} \rceil = 2$
- Adders: $\lceil \frac{4 \times 1}{10} \rceil = 1$
- Not possible to find a schedule with two multipliers

Scheduling

- Two possibilities to find a solution with two multipliers
  - Schedule for more than one sample period: increased flexibility
  - Use cyclic scheduling: "ignore" the start and stop time of the schedule

Multiple Interval Scheduling

- Reasons for scheduling for more than one sample period
  - A schedule obtaining the theoretical bounds on the number of resources can not be found
  - The scheduling time must be longer than the longest execution time

$$T_{\text{schedule}} \geq \max\{T_{\text{exe}}\} \tag{28}$$

Note that storing a variable (shimming delay) has an execution time which is as long as the variable life time

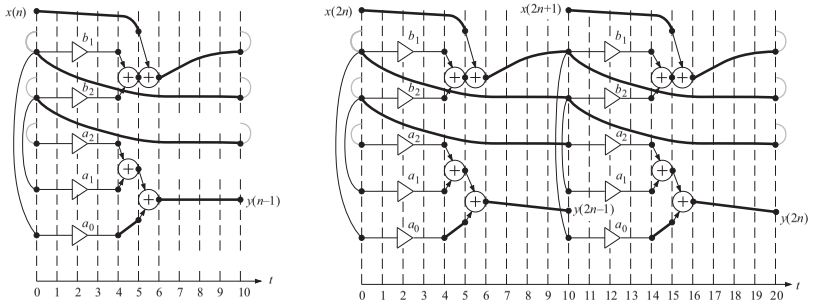
  - The scheduling period is always an integer multiple of the sample period

$$T_{\text{schedule}} = M \times T_{\text{sample}} \tag{29}$$

If there are more than one delay element in the critical loop the resulting minimal sample period may be a fraction

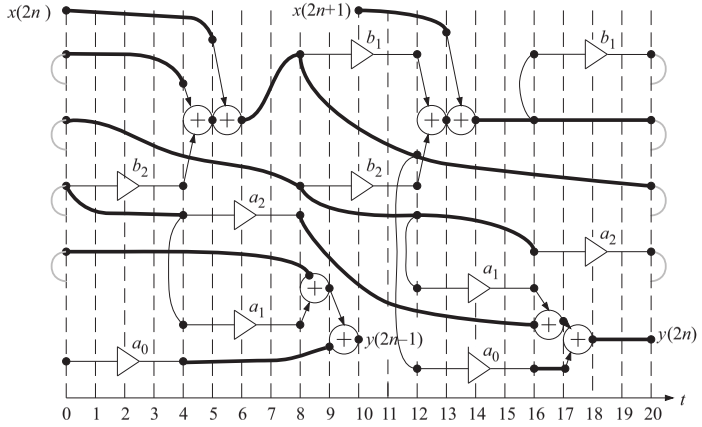
Multiple Interval Scheduling

- Two alternatives to obtain a schedule for more than one sample period
  - Unfold SFG
  - "Copy" computation graph



Multiple Interval Scheduling

- Reschedule



- Now two multipliers are enough
- The longest storage process is still too long but in this case it can be solved by further rescheduling