Application Specific Integrated Circuits for Digital Signal Processing
Lecture 4

Oscar Gustafsson

Today's topic

- Finite word length effects
- DSP algorithms

Finite word length effects

- Focus on fixed-point arithmetic but same things happen in floating-point as well
- Numbers are represented with a limited number of bits
  - Require enough signal-to-noise ratio
  - Must be possible to find a valid transfer function
- Multiplications increase the number of bits
  - Want to quantize (reduce word length)
  - Must quantize in recursive algorithms
- Addition/subtraction may lead to result out of range
  - Need to keep track of maximum signal values to avoid over-/underflow
  - Calculate/estimate possible signal values

Two's complement numbers

- Most common method to represent signed data

$$X = -x_0 + \sum_{i=1}^{W} x_i 2^{-i}, \ -1 \leq X \leq 1 - 2^{-W} \tag{1}$$

- Allows adding/subtracting several numbers in arbitrary order as long as the result is within the correct range

$$(\overbrace{0.11}^{\frac{3}{4}} + \overbrace{0.10}^{\frac{1}{2}}) + (\overbrace{1.10}^{-\frac{1}{2}} + \overbrace{1.00}^{-1}) = \overbrace{1.01}^{-\frac{3}{4}} + \overbrace{0.10}^{\frac{1}{2}} = \overbrace{1.01}^{-\frac{1}{4}} \tag{2}$$

Two's complement numbers

- Largest positive + 1 LSB = Largest negative number

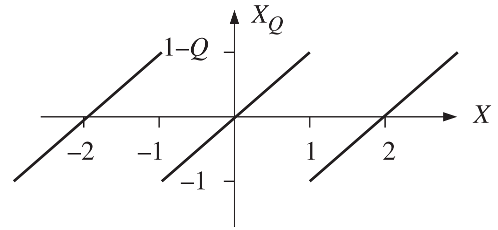
$$\overbrace{0.111\ldots111}^{1-2^{-W}} + \overbrace{0.000\ldots001}^{2^{-W}} = \overbrace{1.000\ldots000}^{-1} \qquad (3)$$

- Largest negative number – 1 LSB = Largest positive number

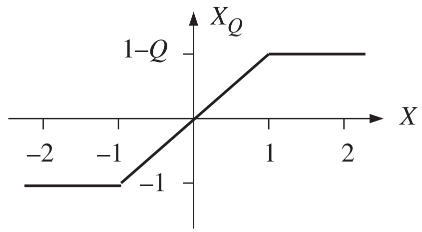
$$\overbrace{1.000\ldots000}^{-1} - \overbrace{1.111\ldots111}^{-2^{-W}} = \overbrace{0.111\ldots111}^{1-2^{-W}} \qquad (4)$$

Overflow

- Overflow and underflow cause distortion and possibly parasitic oscillations
- Two's complement overflow

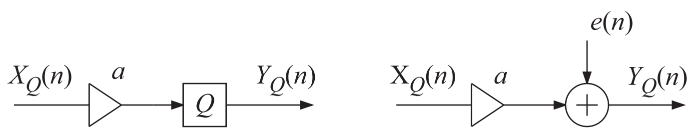


- Saturation arithmetic (clipping)



Data quantization

- Data quantization causes round-off noise and possibly parasitic oscillations
- Data quantization can be viewed as adding a random signal corresponding to the error



- Different types of quantization
- Rounding
  - Proper rounding
  - Add a one to position $W+1$
  - Equivalent to add the bit in position $W+1$ to position W
  - Example: Round 0.0110 and 0.0111 to four bits

$$0.0110 + 0.0001 = 0.011\not{1} \Rightarrow 0.011 \qquad (5)$$

$$0.0111 + 0.0001 = 0.100\not{0} \Rightarrow 0.100 \qquad (6)$$

Data quantization

- Truncation
  - Throw away the unwanted bits
  - Effect on two's complement: rounding towards minus infinity
- Magnitude truncation
  - Rounding towards zero
  - For two's complement: add sign bit to position $W+1$
  - Example: Magnitude truncation of 0.0110 and 1.0111 to four bits

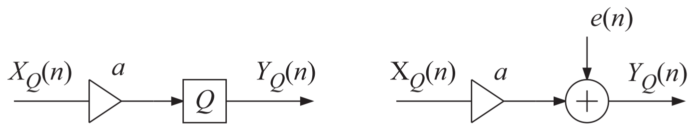
$$0.0110 + \overbrace{0.0000}^{\text{sign bit}} = 0.011\not{0} \Rightarrow 0.011, \ |0.011| \leq |0.0110| \qquad (7)$$

$$1.0111 + \overbrace{0.0001}^{\text{sign bit}} = 1.111\not{0} \Rightarrow 1.111, \ |1.111| \leq |1.0111| \qquad (8)$$

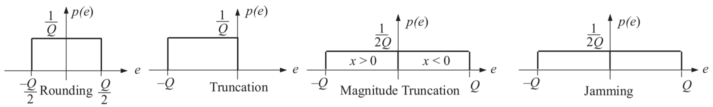
- Jamming (von Neumann rounding)
  - Force LSB to be one

Data quantization and round-off noise

- The round-off noise is a stochastic signal with certain properties



- Error distributions assuming many bits are quantized away ($Q = 2^{-W}$)

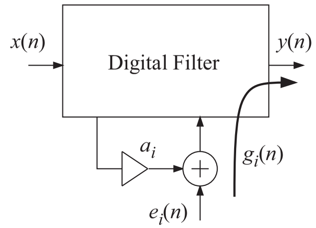


- Means value and variance

Type	Mean, μ	Variance, σ^2
Rounding	0	$\frac{Q^2}{12}$
Truncation	$-\frac{Q}{2}$	$\frac{Q^2}{12}$
Magnitude truncation	correlated with signal	
Jamming	0	$\frac{Q^2}{6}$

Round-off noise

- (Statistics of) total noise can be computed by considering the noise propagation from the sources to the output



- Total mean value

$$\mu_y = \sum_{i=1}^{K} \mu_i \sum_{n=0}^{\infty} g_i(n), \tag{9}$$

- Total noise variance

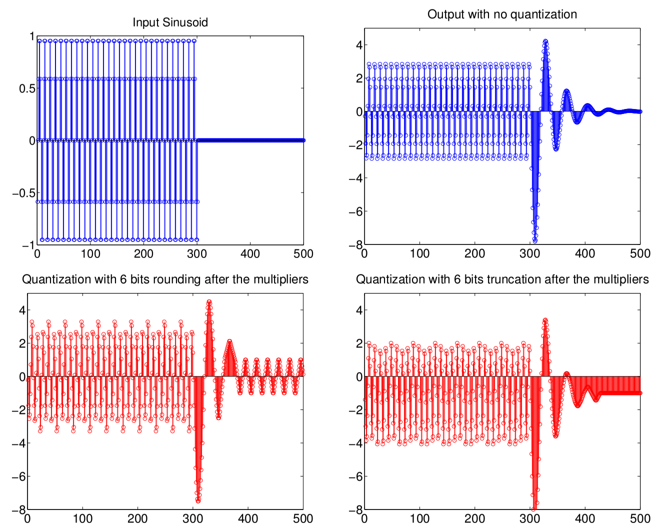
$$\sigma_y^2 = \sum_{i=1}^{K} \sigma_i^2 \sum_{n=0}^{\infty} g_i(n)^2 = \sum_{i=1}^{K} \sigma_i^2 \, \| G_i(z) \|_2^2 = \sum_{i=1}^{K} \frac{\sigma_i^2}{2\pi} \int_{-\pi}^{\pi} \left| G_i(e^{j\omega T}) \right|^2 \mathrm{d}\omega T \tag{10}$$

Limit cycles

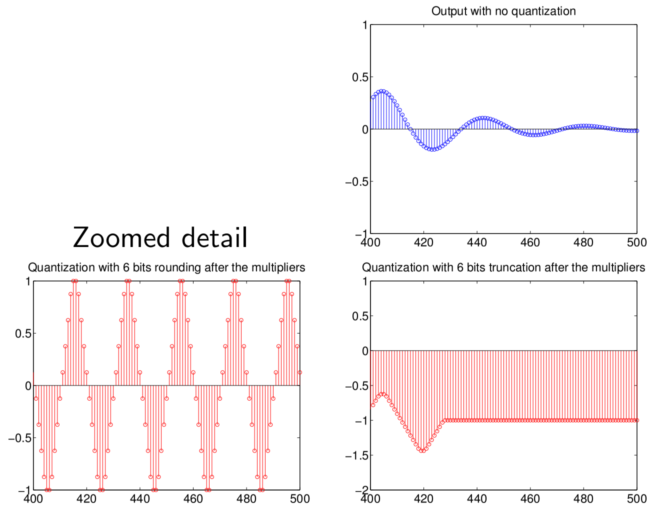
- Input suddenly = 0
  - Output → 0 for stable filter
  - Always the case for non-recursive algorithms
  - Not always the case for all recursive filters
- Example: second-order section

$$H(z) = \frac{1}{1 - \frac{489}{256} z^{-1} + \frac{15}{16} z^{-2}}$$

Limit cycles



Limit cycles



- One possible solution: use longer internal word length and throw away the LSBs

Other parasitic oscillations

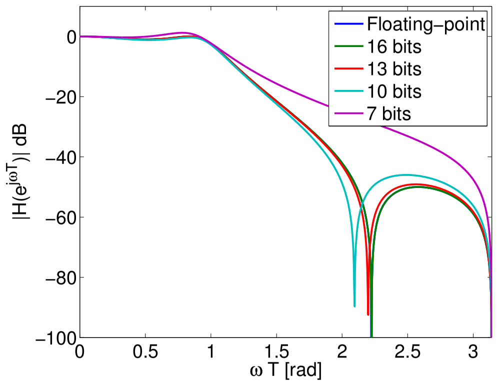
- Constant-input oscillation
  - Triggered by a non-zero DC input level
- Periodic input oscillation
  - Triggered by specific periodic inputs
- Non-observable oscillation
  - Oscillation inside of the filter which does not show up at the output
  - Causes unnecessary switching $\Rightarrow$ power consumption
  - Increases risk of overflow

Wave Digital Filters

- If the signal power (value) is never increased, a WDF based on a passive structure will never be unstable
- Always decrease signal level $\Rightarrow$ magnitude truncation and saturation
- For symmetric two-port adaptor
  - Add one guard bit to the interval word length (MSB side)
  - Use magnitude truncation and saturation at the outputs of the adaptor

Coefficient quantization

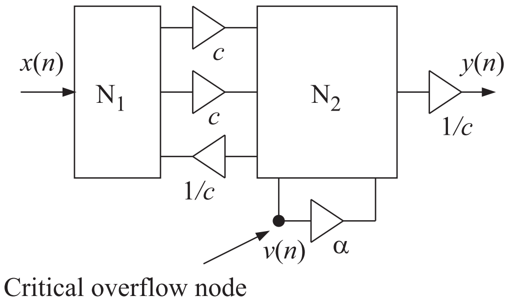
- Quantizing coefficients leads to different coefficients compared to the designed ones
- Static error, so easy to quantify
- Example: Third-order elliptic direct form IIR filter rounded to different number of bits



- Possible (but often hard) to find finite word length coefficients meeting the specifications in a better way than rounding

Scaling

- Signal levels are different in different nodes
- Scale to avoid overflow
- Also scale to utilize numerical range (increase SNR)



- Scaling with 2^s in implementations (to not introduce more quantizations)
- Scale inputs to non-integer multipliers (or use guard bits)
- Additions and subtractions do not need to be scaled for two's complement

Scaling

- Safe scaling – guarantee that overflow never happens
- $f(n)$ impulse response in node v
- $v(n) = f(n) * x(n)$ value in node v
- Assume input $|x(n)| \leq 1$

$$|v(n)| \leq \left| \sum_{k=0}^{\infty} f(k)x(n-k) \right| \leq \sum_{k=0}^{\infty} |f(k)||x(n-k)| \leq \sum_{k=0}^{\infty} |f(k)| \tag{11}$$

- The maximum value of a node is equal to the sum of the absolute values of the impulse response
- The sequence at the input to get this value is ± 1 where the sign is determined by the sign of the impulse response
- Not very likely!

Scaling

- With some knowledge of the statistical propertied of the input signal it is possible to use L_p-norms
- Obtain a scaling value such that it will probably not overflow
- Node overflow is as probable as input overflow

$$\left\| X\left(e^{j\omega T}\right) \right\|_p \equiv \sqrt[p]{\frac{1}{2\pi} \int_{-\pi}^{\pi} \left|X\left(e^{j\omega T}\right)\right|^p \mathrm{d}\omega T} \tag{12}$$

- Meaning of the norms in relation to the absolute value of the Fourier transform in node v, $\left|V\left(e^{j\omega T}\right)\right|$
  - L_1 – Average absolute value
  - L_2 – Related to the signal power, note that

$$\left\| X\left(e^{j\omega T}\right) \right\|_2 = \sqrt[2]{\frac{1}{2\pi} \int_{-\pi}^{\pi} \left|X\left(e^{j\omega T}\right)\right|^2 \mathrm{d}\omega T} = \sqrt{\sum_{-\infty}^{\infty} x(n)^2} \tag{13}$$

  - L_∞ – Maximum value of Fourier transform

Scaling

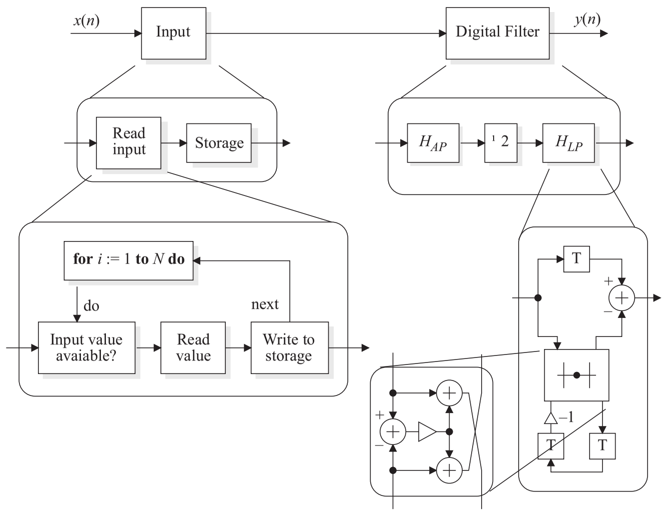
- Signal characterized by L_p, scale by L_q such that $\frac{1}{p} + \frac{1}{q} = 1$

p	q	Signal
∞	1	Wide-band
2	2	Finite signal power
1	∞	Narrow-band

- Selecting power-of-two scaling value will reduce/increase the probability of overflow

DSP System

- Describe using hierarchical processes



DSP algorithms

- Ordered input sequence $x(n)$ mapped to ordered output sequence $y(n)$ using computational rule f

$$x(n) \rightarrow y(n), y(n) = f(x(n)) \tag{14}$$

- Three parts define the output sequence
  - Operation sequence
  - Finite word length properties (word length and quantization)
  - Input data

DSP algorithms

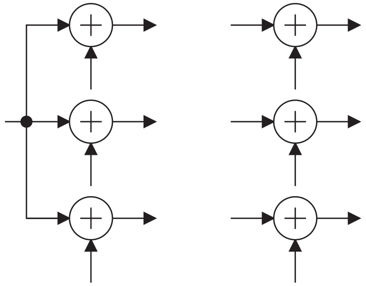
- Iterative processing
  - Continuous stream of data
  - The order of every sample is important
  - Example: digital filter
- Block processing
  - Processing separate blocks
  - The sample order is only important within a block, not among the blocks
  - Example: DFT/FFT, DCT
- Mainly consider iterative processing algorithms
- Often assumed that there is no start or stop time leading to that there will always be a next sample

DSP algorithms

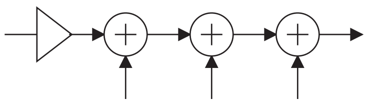
- Characterized by
  - High input and output data rates
  - Input and output values synchronized with the sample period
  - Sequence of operations is data independent
  - Algorithm executed periodically
  - Hard-real time operation: deadline equal to sample period
  - May look like a simple combination of simple operations but often complex theory: speech processing, recursive algorithm stability

Precedence graphs

- Describes the order and dependence of operations
- Parallel algorithm: no precedence between operations



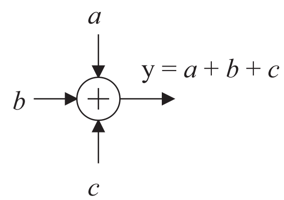
- Sequential algorithm: one precedent and one succedent operation

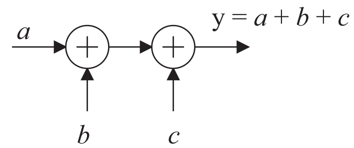


- Almost none completely parallel or sequential algorithm

Constraints to obtain precedence graph

- Sequentially computable algorithm
  - Every directed loop contains at least one delay element
  - No delay free loops
- Fully specified signal flow graphs
  - Algorithm described in operations that will be implemented
  - In most cases: ordering of additions

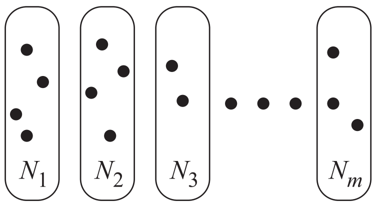




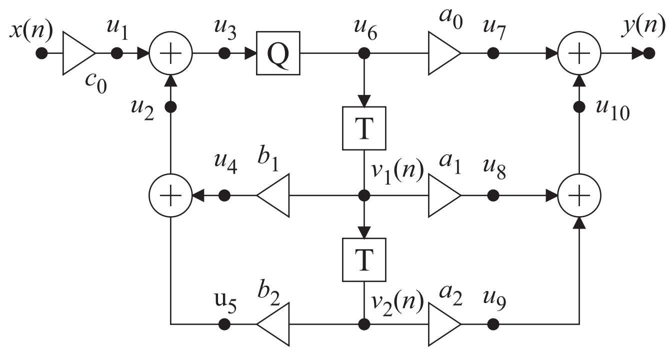
  - Usually not important from algorithmic point of view
  - Important from computational point of view

SFG in precedence form

- Objective: Derive sets of nodes so that the nodes in one set are computable in parallel and sets of nodes are sequentially computable

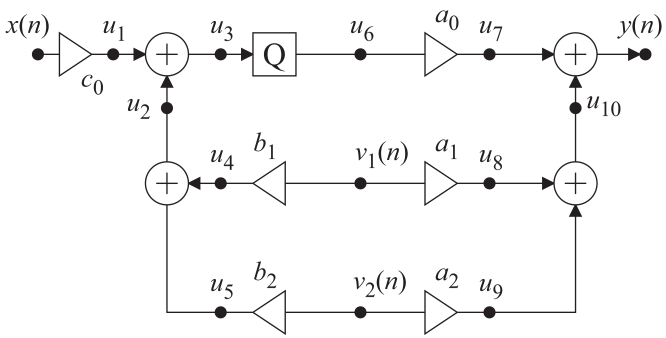


Example



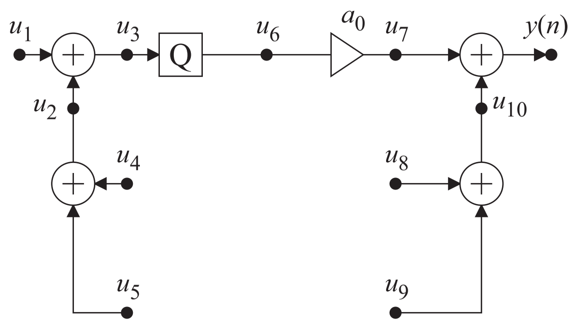
- Remove delay elements

Example



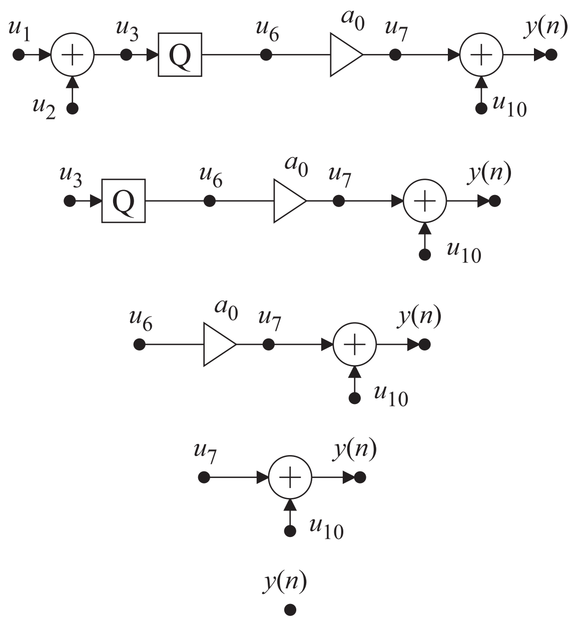
- Put all determined nodes in the first node set (inputs and delay element outputs)
- Remove operations which inputs are determined

Example

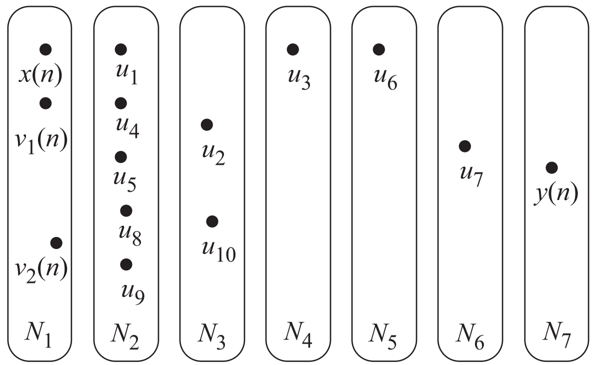


- Put all determined nodes in the second node set
- Remove operations which inputs are determined

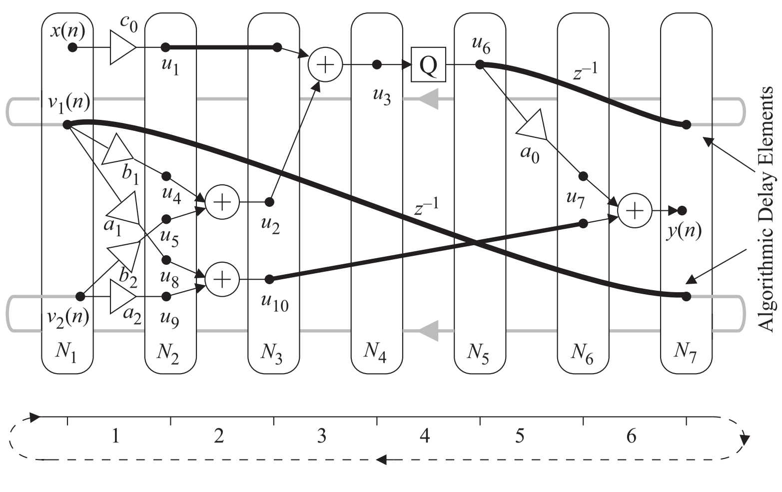
Example



Result

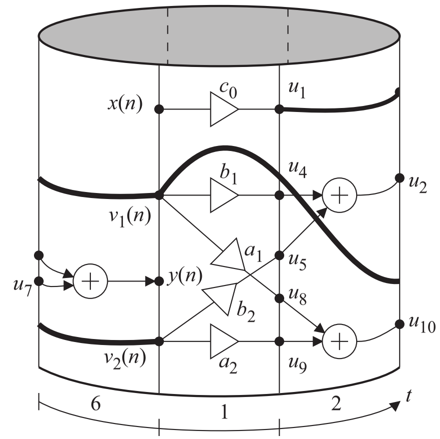


Result

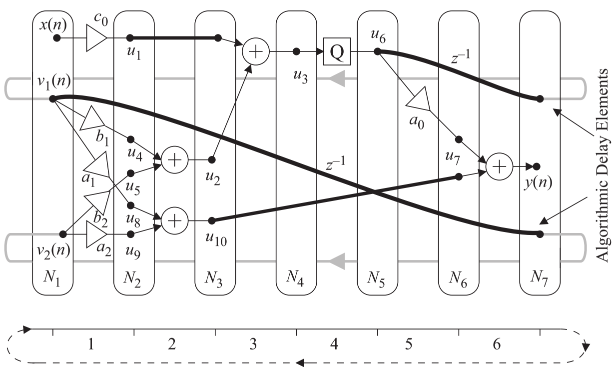


Cylindrical view

Continuous operation leads to that we can see the precedence graph as drawn on a cylinder



Difference equations in computable order



$$\left.\begin{aligned} u_1 &= c_0 x \\ u_4 &= b_1 v_1 \\ u_5 &= b_2 v_2 \\ u_8 &= a_1 v_1 \\ u_9 &= a_2 v_2 \end{aligned}\right\} N_2$$

$$\left.\begin{aligned} u_2 &= u_4 + u_5 \\ u_{10} &= u_8 + u_9 \end{aligned}\right\} N_3$$

$$u_3 = u_1 + u_2 \} N_4$$

$$u_6 = Q[u_3] \} N_5$$

$$u_7 = a_0 u_6 \} N_6$$

$$y = u_7 + u_{10} \} N_7$$

Register updating

- For software it is important to update the state variables in a correct order to avoid overwriting
- Extract delay elements
- Update from the last delay element

  $v_2 = v_1 \} Step_1$

  $v_1 = u_6 \} Step_2$

Difference equation simplification

- It is sometimes possible to merge difference equations resulting in fewer lines
  $u_2 = b_1 v_1 + b_2 v_2$
- In order to avoid redundant computations and numerical issues the following nodes must be computed explicitly
  - Nodes with more than one outgoing branch
  - Output values
  - Register values
  - Inputs to non-linear operations, e.g. quantization