

$$\sum_{k=0}^{n/2} (m'_{2k-1} + k_{2k-1})(m'_{2k} + k_{2k})x^k$$

ECE 699—Digital Signal Processing Hardware Implementations

Lecture 3

FIR Filters and Pipelining (2)

2/4/09

1

Outline

- Fixed-Point Arithmetic in Matlab
- FIR Filters and Pipelining Structures
 - 1) Direct Form FIR Filters
 - 2) Linear-Phase FIR Filters
 - 3) Transpose / Data Broadcast FIR Filters
 - 4) Pipelined FIR Filters
 - 5) Parallel FIR Filters
 - 6) Fast Parallel FIR Filters (Duhamel)
 - 7) Serial/Multi-Cycle FIR Filters
- Implementation issues (time-permitting)
 - Scaling
 - Word growth
 - Carry Save Arithmetic
 - Canonic Signed Digit Filters

2

Reading

- FIR Filters
 - Parhi, VLSI Digital Signal Processing Systems
 - Chapter 3
 - Chapter 9 (Sections 9.1 – 9.2)

Fixed-Point Arithmetic in Matlab

Review: Truncation vs. Round-to-Nearest

S7.5 to S5.3 quantization

ROUND-TO-NEAREST

$$\begin{array}{r} 00.01110 \\ +1 \\ \hline 00.100 \end{array} \quad \begin{array}{r} 10.00110 \\ +1 \\ \hline 10.010 \end{array} \quad \begin{array}{r} 11.01000 \\ +0 \\ \hline 11.010 \end{array}$$

TRUNCATION

$$\begin{array}{r} 00.01110 \\ \hline 00.011 \end{array} \quad \begin{array}{r} 10.00110 \\ \hline 10.001 \end{array} \quad \begin{array}{r} 11.01000 \\ \hline 11.010 \end{array}$$

5

Quantization: Floating to Fixed Point

- Quantize a floating point value to a fixed point value $S_{\text{inf}.L}$, where inf = infinite number of integer bits (hence infinite total bits)
 - Obviously not infinite integer bits, but used to denote fact that we do not take into account integer bits in this calculation
 - Matlab does not have "two's complement" overflow built in. You must force Matlab to overflow/wraparound. More on this later.
- Matlab
 - L = fractional bits desired in fixed point number
 - A_{flp} = floating point signal
 - $A_{\text{fxp}} = \text{floor}(A_{\text{flp}} \cdot 2^L) / 2^L$; → **truncation**
 - $A_{\text{fxp}} = \text{floor}(A_{\text{flp}} \cdot 2^L + 0.5) / 2^L$; → **round to nearest**
- These exactly represent two's complement truncation and rounding in hardware/VHDL
- In Matlab A_{fxp} is still a floating-point number
 - **To be precise, it is a floating-point number modeling a fixed-point number**

6

Example: Truncation

```
>> A_flp = 1.34933434;  
>> L=2;  
>> A_fxp = floor(A_flp*2^L)/2^L  
  
A_fxp =  
  
                1.25  
  
>> L=4;  
>> A_fxp = floor(A_flp*2^L)/2^L  
  
A_fxp =  
  
                1.3125  
  
>> L=6;  
>> A_fxp = floor(A_flp*2^L)/2^L  
  
A_fxp =  
  
                1.34375
```

7

Example: Round to Nearest

```
>> A_flp = 1.34933434;  
>> L=2;  
>> A_fxp = floor(A_flp*2^L+0.5)/2^L  
  
A_fxp =  
  
                1.25  
  
>> L=4;  
>> A_fxp = floor(A_flp*2^L+0.5)/2^L  
  
A_fxp =  
  
                1.375  
  
>> L=6;  
>> A_fxp = floor(A_flp*2^L+0.5)/2^L  
  
A_fxp =  
  
                1.34375
```

8

Quantization: Fixed Point to Fixed Point

- Quantize a fixed point value $S_{inf.L1}$ to a fixed point value $S_{inf.L2}$, where inf = infinite number of integer bits (hence infinite total bits)
 - Obviously not infinite, but used to denote fact that we do not take into account integer bits
- Matlab
 - A_{fxp1} = fixed point signal with $L1$ frac bits
 - $L2$ = number of fractional bits of quantized result
 - $A_{fxp2} = \text{floor}(A_{fxp1} * 2^{L2}) / 2^{L2}$; → **truncation**
 - $A_{fxp2} = \text{floor}(A_{fxp1} * 2^{L2} + 0.5) / 2^{L2}$; → **round to nearest**
- Looks the same as for floating point conversion
 - **No dependence on $L1$ (as long as $L1 \geq L2$)**

9

Example: Rounding

```
>> A_fxp1 = 1.34375; % L1 = 6
>> L2 = 6;
>> A_fxp2 = floor(A_fxp1*2^L2+0.5)/2^L2

A_fxp2 =

           1.34375

>> L2 = 4;
>> A_fxp2 = floor(A_fxp1*2^L2+0.5)/2^L2

A_fxp2 =

           1.375
```

10

Converting Matlab "Fixed Point" to String of Bits

```
% convert Matlab "fixed-point" number (actually it is a floating point number) to string
of bits
if A_fxp < 0
    A_fxp_bits = dec2bin((2^K+A_fxp)*2^L,N)
else
    A_fxp_bits = dec2bin(A_fxp*2^L,N)
end
```

Example:

```
>> A_fxp = 1.34375; N = 8; L = 6; K = N - L;
A_fxp_bits =

01010110

>> A_fxp = -1.34375; N = 8; L = 6; K = N - L;
A_fxp_bits =

10101010
```

11

Converting Matlab String of Bits to "Fixed Point"

```
% convert string of bits to Matlab "fixed-point" number (actually it is a floating point
number)
if A_fxp_bits(1) == '0' % i.e. MSB = 0
    A_fxp_conv = bin2dec(A_fxp_bits)/2^L
else
    A_fxp_conv = bin2dec(A_fxp_bits)/2^L - 2^K
end
```

Results:

```
>> A_fxp_bits = '01010110'; N = 8; L = 6; K = N - L;
A_fxp_conv =

    1.34375

>> A_fxp_bits = '10101010'; N = 8; L = 6; K = N - L;
A_fxp_conv =

   -1.34375
```

12

Wraparound

- All examples thus far assumed there is no wraparound error
- Example how to check wraparound error
 - $A = S4.3, B = S4.3$
 - Steps:
 - Multiply $A * B$ to produce Sinf.6 number (i.e. S8.6)
 - Round to create Sinf.3 number (i.e. S5.3)
 - Check for wraparound to create hardware-accurate S4.3
- This perfectly models removing MSBs in VHDL

13

Wraparound Example

Code:

```
% multiplication example
A = -1; B = 0.875; N = 4; L = 3; K=N-L;
C = A * B; % compute multiplication to produce C = Sinf.6 number
C_quant = floor(C*2^L + 0.5)/2^L; % round to C_quant = Sinf.3 number
%check wraparound
C_quant_wrap = C_quant;
while C_quant_wrap < -(2^(K-1))
    C_quant_wrap = C_quant_wrap + 2^K;
end
while C_quant_wrap > (2^(K-1) - 2^-L)
    C_quant_wrap = C_quant_wrap - 2^K;
end
```

Results:

```
>> C_quant
C_quant =
           -0.875
>> C_quant_wrap
C_quant_wrap =
           -0.875
```

14

Wraparound Example

Results for A = 0.875, B = 0.875:

```
>> C_quant
C_quant =
           0.75

>> C_quant_wrap
C_quant_wrap =
           0.75
```

Results for A = -1, B = -1:

```
>> C_quant
C_quant =
           1

>> C_quant_wrap
C_quant_wrap =
          -1
```

15

$$\sum_{k=0}^{n/2} (m'_{2k-1} + k_{2k-1})(m'_{2k} + k_{2k})x^{-k}$$

FIR Filters

16

FIR Filter Difference Equation

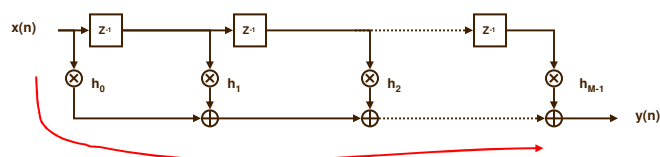
- FIR filter defined by difference equation

$$y(n) = \sum_{i=0}^{M-1} h(i) \cdot x(n-i) = h(n) * x(n)$$

- FIR = finite impulse response
- M-tap filter
 - M "taps" or coefficients
- Often $h(i)$ written as h_i
- Different ways of implementing FIR filter in hardware

17

1) Direct Form FIR Filters



- M-tap FIR filter in direct form
- Critical path:
 - T_A = delay through adder
 - T_M = delay through multiplier
 - Critical path delay: $1 T_M + (M-1) T_A$
- Area:
 - M-1 registers
 - M multipliers
 - M-1 adders
- Latency:
 - Latency is number of cycles between $x(0)$ and $y(0)$, $x(1)$ and $y(1)$, etc.
 - 0 cycles latency
- Arithmetic complexity of M-tap filter modeled as:
 - M multiplications/sample + M-1 adds/sample

18

2) Linear Phase FIR Filters

- Linear phase filter occurs when $h(n) = \pm h(M-1-n)$. M can be odd or even.
- Linear phase filters are used when constant group delay is needed
- Linear phase structures can be designed to save area
- Example: M even
 - Critical path:
 - T_A = delay through adder
 - T_M = delay through multiplier
 - Critical path delay: $1 T_M + (M/2) T_A$
 - Area:
 - $M-1$ registers
 - $M/2$ multipliers
 - $M-1$ adders

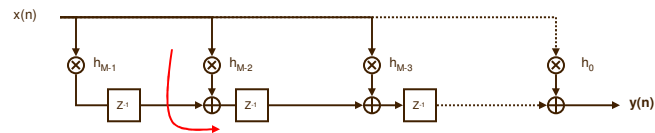
19

3) Direct Form Transpose Filters

- FIR filter can be decomposed into a signal flow graph
 - Nodes
 - Edges
- SFG transposition rule: "Reversing the direction of an SFG and interchanging the input and output ports preserves the functionality of the system."
- Transposition to direct form filter results in direct form transpose filter, also called data broadcast structure

20

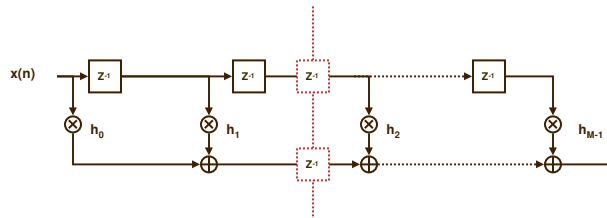
Direct Form Transpose Filters



- Use a signal flow graph reversal to reduce the critical path \rightarrow transpose structure
- Critical path:
 - Delay: $1 T_M + 1 T_A$
- Area:
 - M-1 registers
 - M multipliers
 - M-1 adders
- Latency:
 - 0 cycles latency
- Arithmetic complexity of M-tap filter modeled as:
 - M multiplications/sample + M-1 adds/sample
- Disadvantages
 - Larger register sizes depending on quantization scheme used
 - Fanout of $x(n)$ can become prohibitive

21

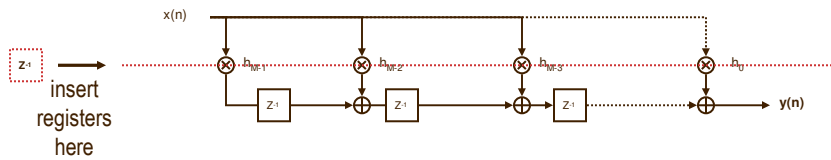
4) Pipelined FIR Filters



- Example: coarse-grain pipelining for direct form filter
- Pipelining generally only valid for feed-forward cutsets of a SFG
 - Feedback structures will be covered later

22

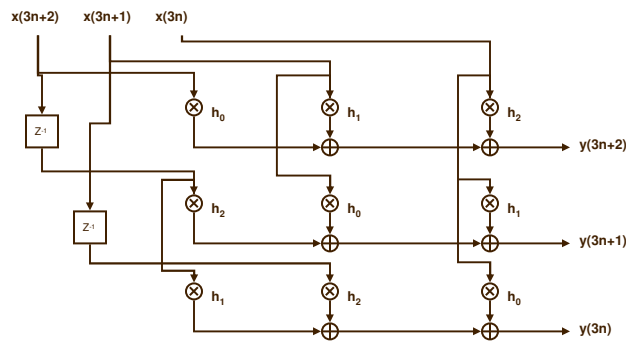
Fine-Grain Pipelining



- Fine-grain pipelining allows for reduction of critical path in transposed structures

23

5) Parallel FIR Filters



- Parallel processing maintains overall sample throughput while reducing clock rate
- Useful: when input/output bottlenecks exist

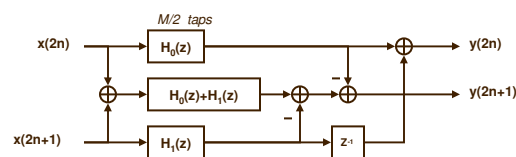
24

Parallel and Pipelining Processing for Low Power in ASICs

- In CMOS circuits, power is proportional to the square of the supply voltage
 - At the output of a CMOS gate, $P = \alpha * C * V_{dd}^2 * f$
 - Alpha = activity factor
 - C = capacitance/load of gate
 - Vdd = supply voltage
 - f = clock frequency
- Reducing supply voltage reduces power consumption dramatically
 - 1 V \rightarrow sample chip power = 10 W
 - .7 V \rightarrow sample chip power = 4.9 W \rightarrow 51% decrease in power from 30% decrease in voltage
- Parallel processing and pipelining can help with low power design

25

6) Fast Parallel FIR Filters



- Direct form and transpose form structures (running at the same rate) with M taps require M multiplications/sample and M-1 adds/sample
- Methods exist to reduce this complexity by parallel processing and subexpression sharing.
- In the 2-parallel structure above, two inputs arrive at half the original clock rate and are processed in parallel by three $\text{ceil}(M/2)$ -tap filters [$\text{ceil}()$ is the ceiling function]
- Arithmetic complexity of the 2-parallel filter is approximately:
 - $3 \times M/2$ multiplications / two samples + $3 \times (M/2 - 1)$ adds / two samples + 4 adds / two samples
 - $= 3/4 M$ multiplications/sample + $(3M/4 + 1/2)$ adds/sample
- If power is dominated by multipliers, 25% power savings over traditional structures!

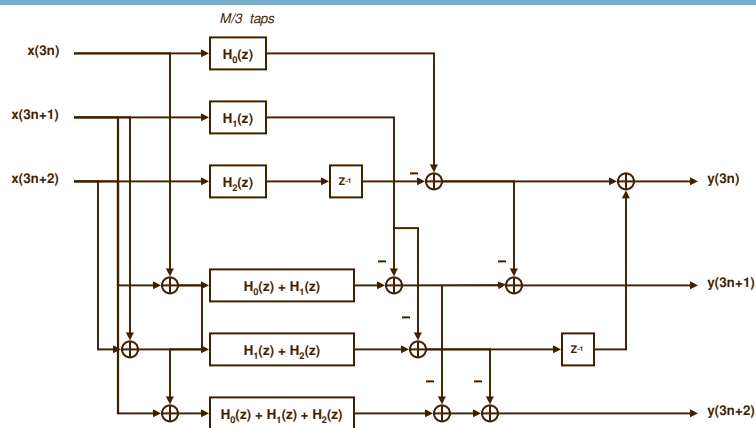
26

Coefficients for 2-parallel filter

- Example for $M = 8$
 - $H(z) = \{h_0, h_1, h_2, h_3, h_4, h_5, h_6, h_7\}$
- Subfilter coefficients obtained by performing a polyphase decomposition by 2. Each subfilter has $M/2 = 4$ coefficients:
 - $H_0(z) = \{h_0, h_2, h_4, h_6\}$
 - $H_1(z) = \{h_1, h_3, h_5, h_7\}$
 - $H_0(z) + H_1(z) = \{h_0+h_1, h_2+h_3, h_4+h_5, h_6+h_7\}$
- May have wordlength growth in $H_0(z) + H_1(z)$ combined coefficient

27

3-Parallel Fast FIR Filter



- In the 3-parallel filter, three inputs arriving at a third of the original rate are processed by six parallel $\text{ceil}(M/3)$ -tap filters
- Arithmetic complexity of the 3-parallel filter is approximately:
 - $2/3 M$ multiplications/sample + $(2/3M + 4/3)$ adds
 - **33% reduction in multiplications/sample**

28

Coefficients of 3-Parallel Filters

- Example for $M = 9$
 - $H(z) = \{h_0, h_1, h_2, h_3, h_4, h_5, h_6, h_7, h_8\}$
- Subfilter coefficients obtained by performing a polyphase decomposition by 3. Each subfilter has $M/3 = 3$ coefficients:
 - $H_0(z) = \{h_0, h_3, h_6\}$
 - $H_1(z) = \{h_1, h_4, h_7\}$
 - $H_2(z) = \{h_2, h_5, h_8\}$
 - $H_0(z) + H_1(z) = \{h_0+h_1, h_3+h_4, h_6+h_7\}$
 - $H_1(z) + H_2(z) = \{h_1+h_2, h_4+h_5, h_7+h_8\}$
 - $H_0(z) + H_1(z) + H_2(z) = \{h_0+h_1+h_2, h_3+h_4+h_5, h_6+h_7+h_8\}$

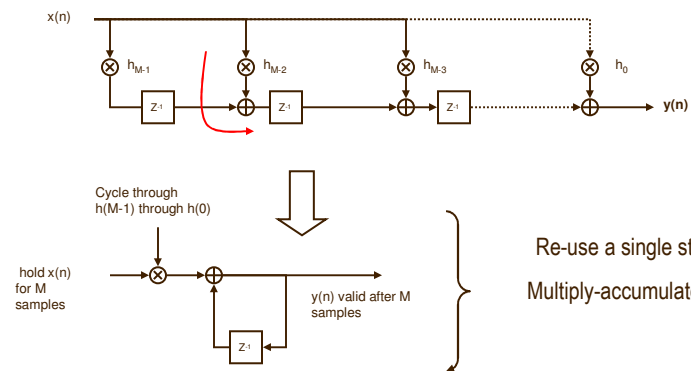
29

Further Parallelism

- These parallel structures introduce issues such as increased area, adder overhead (pre- and post-processing), etc. which eventually become prohibitive as the subsampling rate increases

30

7) Serial / Multi-Cycle



- Trade off area for speed
 - Parallel filter: M multipliers, output ready in one cycle
 - Serial filter: 1 multiplier, output ready in M cycles