VLSI Signal Processing

Lecture 10 Numerical Strength Reduction

Subexpression Elimination

- Sub-expression elimination is a numerical transformation of the constant multiplications that can lead to efficient hardware in terms of area, power, and speed.
- Sub-expression can only be performed on constant multiplications that operate on a common variable.
- It is essentially the process of examining the shift-and-add implementations of the constant multiplications and finding redundant operations.
- Once the redundancies are found, these operations can be performed once and shared among the constant multiplications.

Example

- $a \times x$ and $b \times x$, where a=001101 and b=011011 can be performed as follows:
  - $a \times x = 000100 \times x + 001001 \times x$
  - $b \times x = 010010 \times x + 001001 \times x$
    $= (001001 \times x) << 1 + 001001 \times x$
  - The term $001001 \times x$ is redundant and can be computed only once.
  - The multiplications were implemented using 3 shifts and 3 adds as opposed to 5 shifts and 5 adds.
  - Also note that $b \times x = (2a+1) \times x = 2(a \times x) + x$. Alternately, by computing $a \times x$ first, it also requires 3 shifts and 3 adds.

- Matching terms are redundant !!

Multiple Constant Multiplication (MCM)

- To apply the subexpression elimination to a set of constant multipliers that multiply a common variable.
- The goal is to find the minimum number of shifts and adds, i.e. to find the best match !!
- Iterative matching algorithm
  1. Express each constant in the set using a binary form
  2. Determine the number of nonzero bit-wise matches between all of the constant in the set
  3. Choose the best match
  4. Eliminate the redundancy from the best match. Return the remainder and the redundancy to the set of coefficients
  5. Repeat step 2-4 until no improvement is achieved.

MCM Example

- a=237, b=182, c=93.
- Step 1

Constant	Value	Unsigned
a	237	11101101
b	182	10110110
c	93	01011101

Binary representation of constants

- Step 2, determine the matches among them
  - a v.s. b: 3 matches
  - a v.s. c: 4 matches
  - b v.s. c: 2 matches

MCM Example

- Step 3, the match between a and c is selected.
- Step 4,

Constant	Value	Unsigned
a	237	11101101
b	182	10110110
c	93	01011101

Binary representation of constants

Constant	Unsigned
Rem. of a	10100000
b	10110110
Rem. of c	00010000
Red. of a,c	01001101

Updated set of constants
1st iteration

- Repeat the process

Example MCM

Constant	Unsigned
Rem. of a	00000000
Rem. of b	00010110
Rem. of c	00010000
Red. of a,c	01001101
Red. of Rem a,b	10100000

Redundant terms (computed only once)

Updated set of constants
2nd iteration

- The implementations are as follows:
  a = [01001101 + 10100000]
  b = [00010110 + 10100000]
  c = [00010000 + 10100000]

9 shifts and 9 adds !!

Linear Transformations

- One can apply the iterative matching algorithm to linear transformations.
- General form: $\mathbf{y}_{m\times1} = \mathbf{T}_{m\times n}\ \mathbf{x}_{n\times1}$ $\qquad y_i = \sum_{j=1}^{n} t_{ij}x_j, i = 1,\ldots,m$
- 3 steps
  - To minimize the number of shifts and adds required to compute the product $t_{ij}x_j$ by using the iterative matching algorithm
  - Formation of unique products using the sub-expression found in the 1st step.
  - Final step involves the sharing of adds, which is common among the y_i's. (This step is very similar to MCM problem)

Example

$$T=\begin{bmatrix} 7 & 8 & 2 & 13 \\ 12 & 11 & 7 & 13 \\ 5 & 8 & 2 & 15 \\ 7 & 11 & 7 & 11 \end{bmatrix}$$

- The constants in each column multiply to a common variable. For Example x_1 is multiplied to the set of constants [7, 12, 5, 7].
- Applying iterative matching algorithm the following table is obtained.

Column 1	Column 2	Column 3	Column 4
0101	1000	0010	1001
0010	1011	0111	0100
1100			0010

- Next, the unique products are formed as shown below:

$$p_1 = 0101*x_1,\ p_2 = 0010*x_1,\ p_3 = 1100*x_1$$
$$p_4 = 1000*x_2,\ p_5 = 1011*x_2,$$
$$p_6 = 0010*x_3,\ p_7 = 0111*x_3$$
$$p_8 = 1001*x_4,\ p_9 = 0100*x_4,\ p_{10} = 0010*x_4$$

- Using these products the y_i's are as follows:

$$y_1 = \overset{7}{p_1 + p_2} + \overset{8}{p_4} + \overset{2}{p_6} + \overset{13}{p_8 + p_9};$$
$$y_2 = p_3 + p_5 + p_7 + p_8 + p_9;$$
$$y_3 = p_1 + p_4 + p_6 + p_8 + p_9 + p_{10};$$
$$y_4 = p_1 + p_2 + p_5 + p_7 + p_8 + p_{10};$$

- This step involves sharing of additions which are common to all y_i's. For this each y_i is represented as k bit word ($1 \le k \le 10$), where each of the k products formed after the 2nd step represents a particular bit position. Thus,

$$y_1 = 1101010110,\ y_2 = 0010101110,$$
$$y_3 = 1001010111,\ y_4 = 1100101101.$$

- Applying iterative matching algorithm to reduce the number of additions required for y_i's we get:

$$y_1 = p_2 + (p_1 + p_4 + p_6 + p_8 + p_9);$$
$$y_2 = p_3 + p_9 + (p_5 + p_7 + p_8);$$
$$y_3 = p_{10} + (p_1 + p_4 + p_6 + p_8 + p_9);$$
$$y_4 = p_1 + p_2 + p_{10} + (p_5 + p_7 + p_8);$$

- The total number of additions are reduced from 35 to 20.

Polynomial Evaluation

- One can apply the subexpression elimination to polynomial evaluation
- Suppose we are to evaluate the polynomial

$$x^{13} + x^7 + x^4 + x^2 + x$$

- By directly computation, it requires 22 multiplications
- Subexpression elimination
  - Examining the exponents, 1, 2, 4, 7, and 13, and considering their binary representations
  - Applying sub-expression sharing to the exponents, then

$$x^8(x^4x) + x^2(x^4x) + x^4 + x^2 + x$$

  - The terms x^2, x^4 and x^8 each requires one multiplication:

$$x^2 = x \times x,\ x^4 = x^2 \times x^2,\ x^8 = x^4 \times x^4$$

  - Totally, it requires 6 multiplications

Multiple Polynomials

1. To reduce the number of multiplications required to generate the various powers of x
2. To reduce the number of shifts and adds required to implement the multiplications of the power terms by the constant coefficients

- Example:

$$w(x)=11x^5+3x^4+6x^3+5x,$$
$$y(x)=13x^5+7x^4+11x^3,$$
$$w(x)=7x^5+15x^4+5x^2+7x$$

1. exponent: 1, 2, 3, 4, 5
2. coefficient: (11,13,7), (3, 7, 15), (6,11), (5,7)

Subexpression Sharing

- Example of common sub-expression elimination within a single multiplication :

$$y = 0.10\bar{1}000 10\bar{1} * x.$$

CSD, 2's-complements fixed point rep.

This may be implemented as:

$$y = (x >> 1) - (x >> 3) + (x >> 7) - (x >> 9).$$

Alternatively, this can be implemented as,

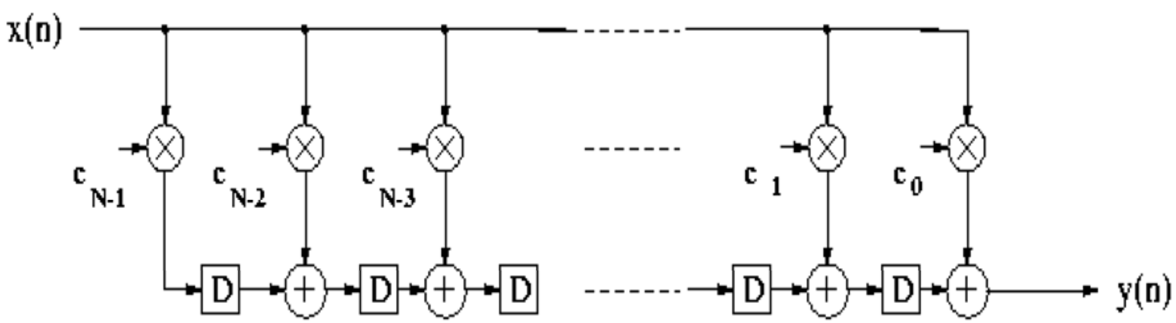
$$x2 = x - (x >> 2)$$

$$Y = (x2 >> 1) + (x2 >> 7)$$

which requires one less addition.

Subexpression Sharing in Filters

- Data broadcast filter architecture:
  - Several constants need to be multiplied to a common variables:



$$y(n) = c_0x(n) + c_1x(n-1) + \dots + c_0x(n-N+1)$$

Steps

- Represent a filter operation by a table (matrix) $\{x_{ij}\}$, where the rows are indexed by delay i and the columns by shift j, i.e., the row i is the coefficient c_i for the term x(n-i), and the column 0 in row i is the msb of c_i and column W-1 in row i is the lsb of c_i , where W is the word length.
- The row and column indexing starts at 0.
- The entries are 0 or 1 if 2's complement representation is used and $\{1, 0, \bar{1}\}$ if CSD is used.
- A non-zero entry in row i and column j represents x(n-i) >> j. It is to be added or subtracted according to whether the entry is +1 or -1.

DEPT. OF ELECTRONICS ENGINEERING & INST. OF ELECTRONICS NCTU

Example: 3-Tap FIR Filter

$$y(n) = 1.000\bar{1}00000 * x(n) + 0.\bar{1}0\bar{1}010010 * x(n-1) + 0.00010000\bar{1} * x(n-2)$$

CSD Coeff.

msb ... lsb

c_0	1				-1					
c_1		-1		-1		1			1	
c_2					1					-1

7 adds. !!

This filter has 8 non-zero terms and thus requires 7 additions. But, the sub-expressions x1 + x1[-1] >> 1 occurs 4 times in shifted and delayed forms by various amounts as circled. So, the filter requires 4 adds.

$$x2 = \boxed{x1 - x1[-1] >> 1}$$

$$y = x2 - (x2 >> 4) - (x2[-1] >> 3) + (x2[-1] >> 8)$$

An alternative realization is :

$$x2 = \boxed{x1 - (x1 >> 4) - (x1[-1] >> 3) + (x1[-1] >> 8)}$$

$$y = x2 - (x2[-1] >> 1).$$

Example: 4-Tap FIR Filter

$$y(n) = \bar{1}.01010000010 * x(n) + 0.\bar{1}000\bar{1}0\bar{1}0\bar{1}0\bar{1} * x(n-1)$$
$$+ 0.\bar{1}0010000010 * x(n-2) + 1.0000010\bar{1}000 * x(n-4)$$

The substructure matching procedure for this design is as follows:

- Start with the table containing the coefficients of the FIR filter. An entry with absolute value of 1 in this table denotes add or subtract of x1. Identify the best sub-expression of size 2.

The number of occurrences

-1		1		1						1	
	-1				-1		-1		-1		-1
	-1			1						1	
1						1		-1			

- Remove each occurrence of each sub-expression and replace it by a value of 2 or -2 in place of the first (row major) of the 2 terms making up the sub-expression.

-1		2		1						2	
					-2		-1				-2
	-2										
						1		-1			

	1
-1	

- Record the definition of the sub-expression. This may require a negative value of shift which will be taken care of later.

x2 = x1 - x1[-1] >> (-1)

- Continue by finding more subexpressions until done.

-1		2		1						2	
					-2		-1				-2
	-2										
						1		-1			

-1		3								2	
					-3						-2
	-2										
						1		-1			

$x3 = x2 + x1 >> 2$

- Write out the complete definition of the filter.

$$x2 = x1 - x1[-1] >> (-1)$$
$$x3 = x2 + x1 >> 2$$
$$y = -x1 + x3 >> 2 + x2 >> 10 - x3[-1] >> 5 - x2[-1] >> 11$$
$$-x2[-2] >> 1 + x1[-3] >> 6 - x1[-3] >> 8.$$

	0	1	2	3	4	5	6	7	8	9	10	11
0	-1		3								2	
-1						-3						-2
-2		-2										
-3							1		-1			

msb lsb

- If any sub-expression definition involves negative shift, then modify the definition and subsequent uses of that variable to remove the negative shift as shown below:

$x2 = x1 - x1[-1] \gg (-1)$ $\quad\quad$ $x2 = x1 \gg 1 - x1[-1]$

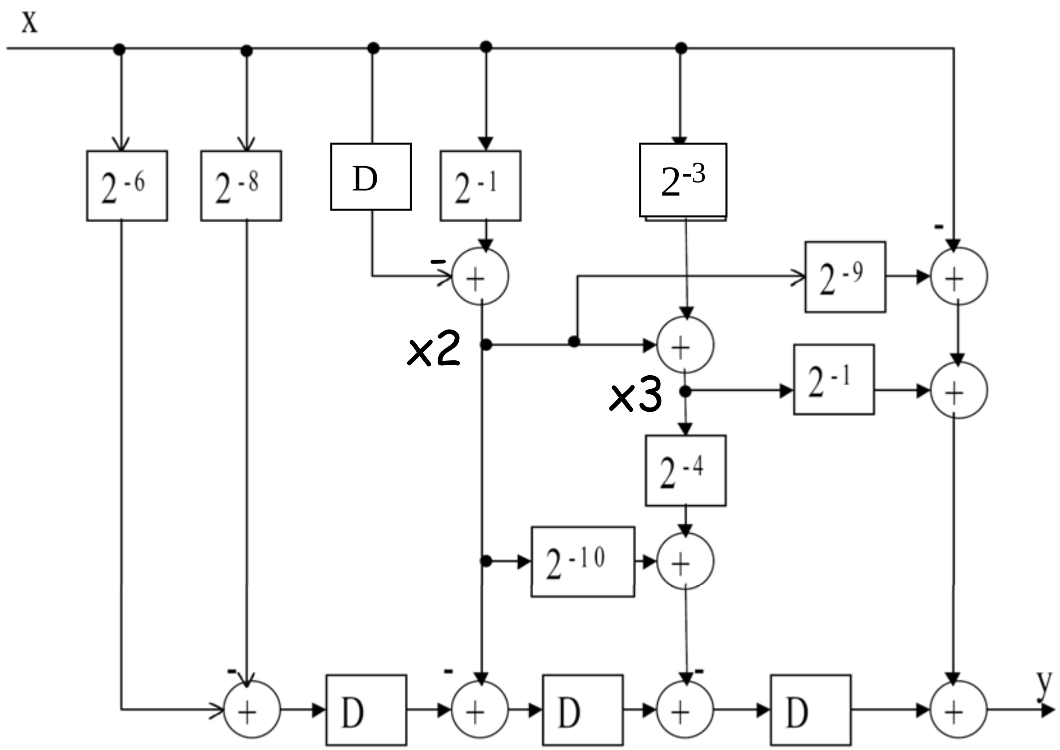
	1
-1	

$$x2 = x1 \gg 1 - x1[-1]$$
$$x3 = x2 + x1 \gg 3$$
$$y = -x1 + x3 \gg 1 + x2 \gg 9 - x3[-1] \gg 4 - x2[-1] \gg 10$$
$$- x2[-2] + x1[-3] \gg 6 - x1[-3] \gg 8.$$

$$x2 = x1 >> 1 - x1[-1]$$

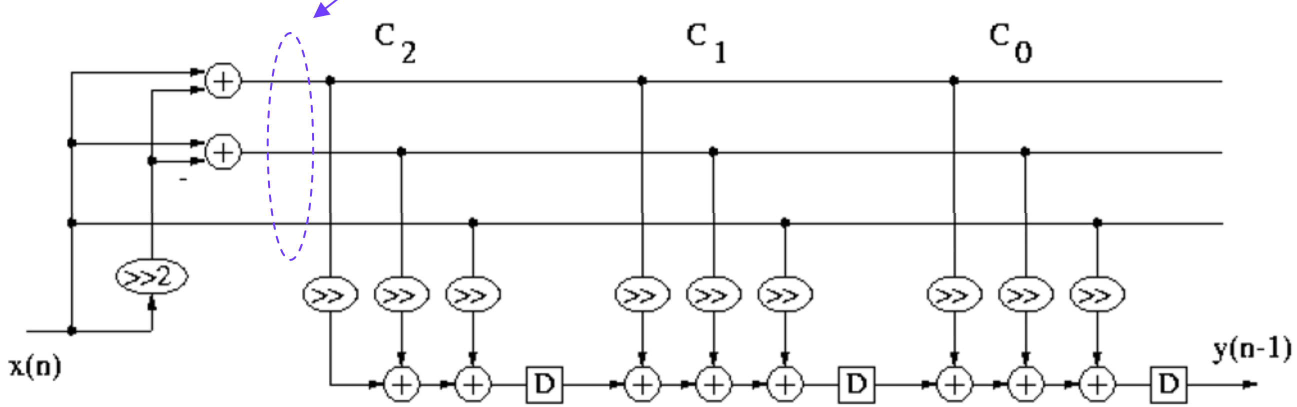
$$x3 = x2 + x1 >> 3$$

$$y = -x1 + x3 >> 1 + x2 >> 9 - x3[-1] >> 4 - x2[-1] >> 10$$
$$- x2[-2] + x1[-3] >> 6 - x1[-3] >> 8.$$

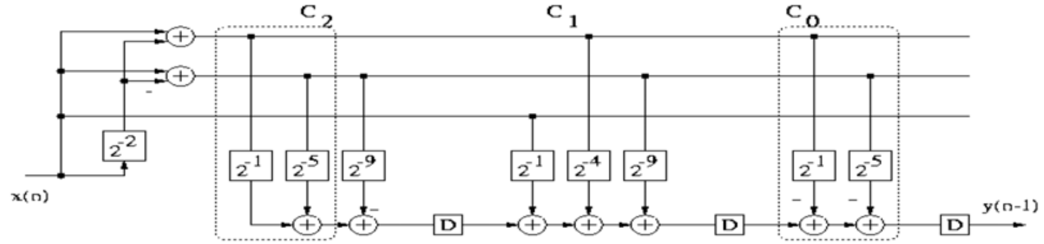


Remarks

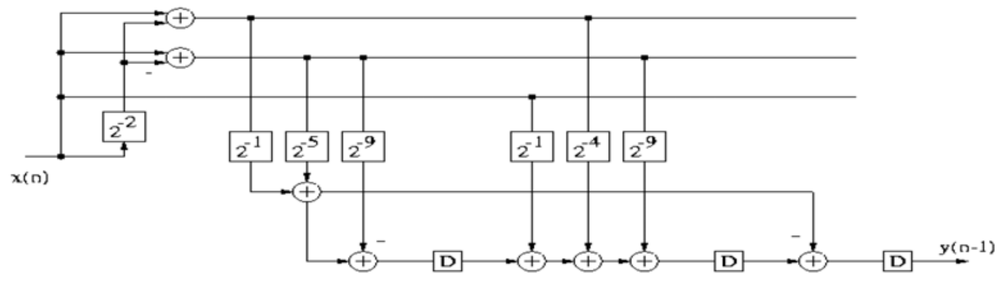
- Digital filters can be implemented with less hardware using CSD coefficients.
- In CSD, 2 most common subexpressions are x-x>>2 and x+x>>2, corresponding to sequences $10\bar{1}$ and 101, respectively.
- Fact: all CSD coefficients can be built using 3 fundamental subexpressions: $10\bar{1}$, 101, and 1.
- Example



Using 2 Most Common Subexpression in CSD Representations



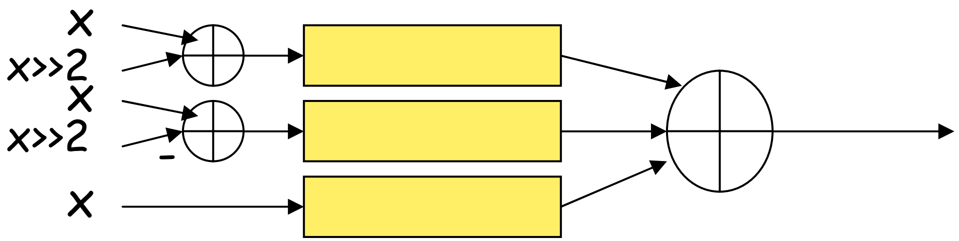
$c_2 = 0.\underline{101}0\underline{10\bar{1}}0\underline{10\bar{1}}$ $\quad c_1 = 0.100\underline{101}00\underline{10\bar{1}}$ $\quad c_0 = 0.\underline{10\bar{1}}0\underline{10\bar{1}}0000$



(b)

3-tap FIR filter with coefficients $c_2 = 0.101010\bar{1}010\bar{1}$, $c_1 = 0.1001010010\bar{1}$ and $c_0 = 0.\bar{1}0\bar{1}0\bar{1}010000$. 2 additions in the dotted square in (a) are shared in (b). Filter requires only 7 additions and 7 shifts as opposed to 12 adds and 12 shifts in standard multiplierless implementation.

Remark



- An alternative layout is shown above.
- In this case, best results will be achieved if three data paths are to some extent balanced.
- Balance ➔ the number of adds (or adders) in each stages are (statistically) minimum ➔ this achieves the maximum clock rate of the circuit
- Note:
  - The number of terms in the x-data-path is on the average twice as many as in the (x+x>>2) and (x-x>>2) paths
  - This inequality can be redressed by swapping x terms for (x+x>>2) and (x-x>>2) terms.
  - Example: $1001 \rightarrow \underline{101\bar{1}}, \quad 10001 \rightarrow \underline{10\bar{1}00} + \underline{00101}$