Chapter 15: Numerical Strength Reduction

Keshab K. Parhi

- 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.
- Example: $a \times x$ and $b \times x$, where a = 001101 and

  b = 011011 can be performed as follows:
  - $a \times x = 000100 \times x + 001001 \times x$
  - $b \times x = 010010 \times x + 001001 \times x = (001001 \times x) \ll 1 + (001001 \times x)$.
  - The term $001001 \times x$ needs to be computed only once.
  - So, multiplications were implemented using 3 shifts and 3 adds as opposed to 5 shifts and 5 adds.

Multiple Constant Multiplication(MCM)

The algorithm for MCM uses an iterative matching process that consists of the following steps:

- Express each constant in the set using a binary format (such as signed, unsigned, 2's complement representation).
- Determine the number of bit-wise matches (non-zero bits) between all of the constants in the set.
- Choose the best match.
- Eliminate the redundancy from the best match. Return the remainders and the redundancy to the set of coefficients.
- Repeat Steps 2-4 until no improvement is achieved.

Example:

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

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

Updated set of constants
2nd iteration

Linear Transformations

- A general form of linear transformation is given as:

y =T*x

where, T is an m by n matrix, y is length-m vector and x is a length-n vector. It can also be written as:

$$y_i = \sum_{j=1}^{n} t_{ij} x_j, i = 1, \ldots, m$$

- The following steps are followed:
  - Minimize the number of shifts and adds required to compute the products $t_{ij}x_j$ 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 additions, which is common among the y_i's. This step is very similar to the 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:

$$y_1 = p_1 + p_2 + p_4 + p_6 + p_8 + p_9;$$

$$y_2 = p_3 + p_5 + p_7 + p_8 + p_9;$$

$$y_3 = p_1 + p_4 + p_6 + p_8 + p_9 + p_{10};$$

$$y_4 = p_1 + p_2 + p_5 + p_7 + p_8 + p_{10};$$

- 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

Evaluating the polynomial:

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

- Without considering the redundancies this polynomial evaluation requires 22 multiplications.
- Examining the exponents and considering their binary representations:

$$1 = 0001,\ 2 = 0010,\ 4 = 0100,\ 7 = 0111,\ 13 = 1101.$$

- x^7 can be considered as $x^4 \times x^2 \times x^1$. Applying sub-expression sharing to the exponents the polynomial can be evaluated as follows:

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

- The terms x^2, x^4 and x^8 each require one multiplication as shown below:

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

- Thus, we require 6 instead of 22 multiplications.

Sub-expression Sharing in Digital Filters

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

$$y = 0.10\bar{1}00010\bar{1} * x.$$

This may be implemented as:

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

Alternatively, this can be implemented as,

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

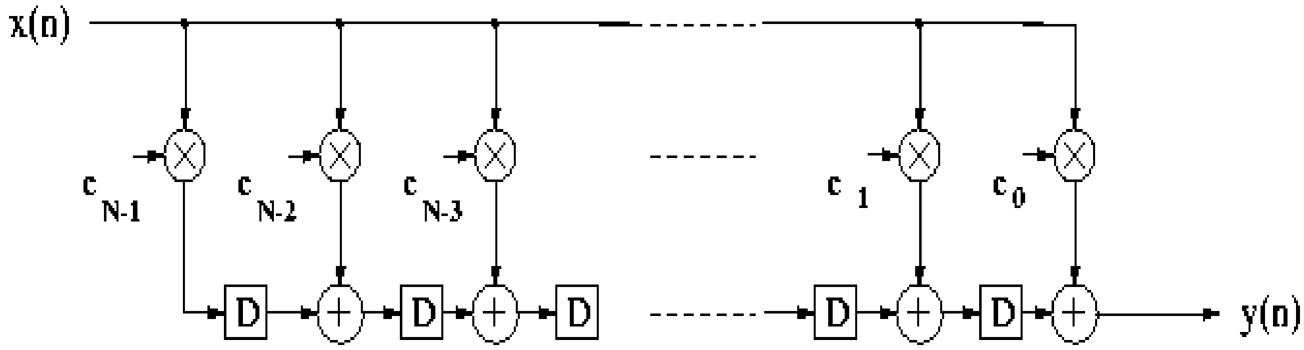
$$Y = (x2 >> 1) + (x2 >> 7)$$

which requires one less addition.

- In order to realize the sub-expression elimination transformation, the N-tap FIR filter:

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

must be implemented using transposed direct-form structure also called data-broadcast filter structure as shown below:



- 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.

Example:

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

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

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.

$$x2 = x1 – x1[-1] >> 1$$

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

An alternative realization is :

$$x2 = x1 – (x1 >> 4) – (x1[-1] >> 3) + (x1[-1] >> 8)$$

$$y = x2 – (x2[-1] >> 1).$$

Example:

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

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.

-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		-1		-2
	-2										
						1		-1			

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

x3 = x1 – x1[-1] >> (-1)

- Continue by finding more sub-expressions until done.

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

5. Write out the complete definition of the filter.

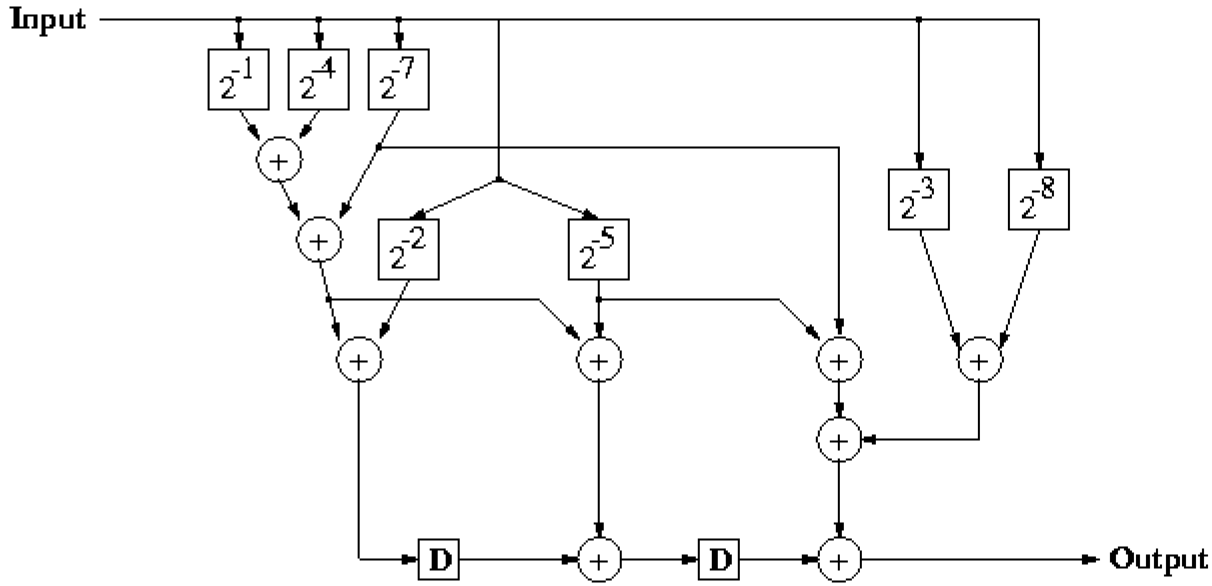
$$x2 = x1 - x1[-1] \gg (-1)$$
$$x3 = x2 + x1 \gg 2$$
$$y = -x1 + x3 \gg 2 + x2 \gg 10 - x3[-1] \gg 5 - x2[-1] \gg 11$$
$$-x2[-2] \gg 1 + x1[-3] \gg 6 - x1[-3] \gg 8.$$

- 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:

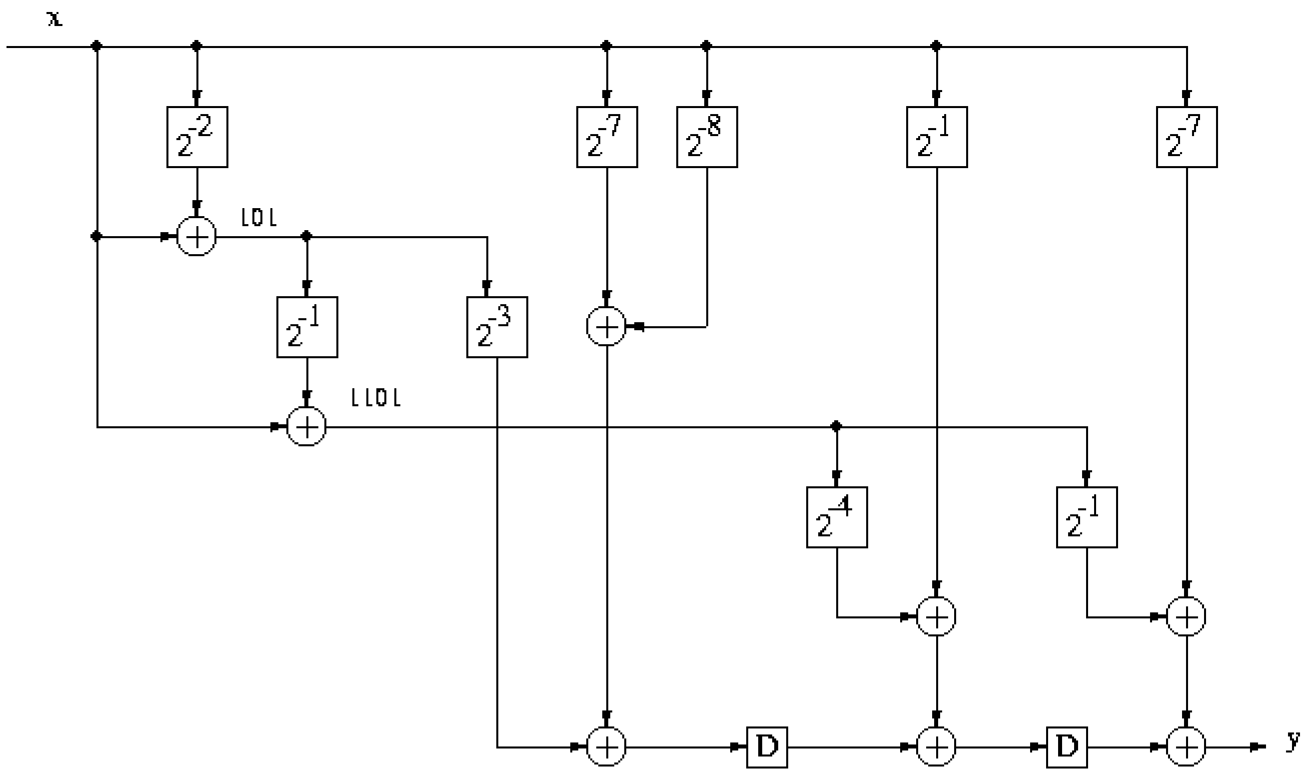
$$x2 = x1 >> 1 - x1[-1]$$

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

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



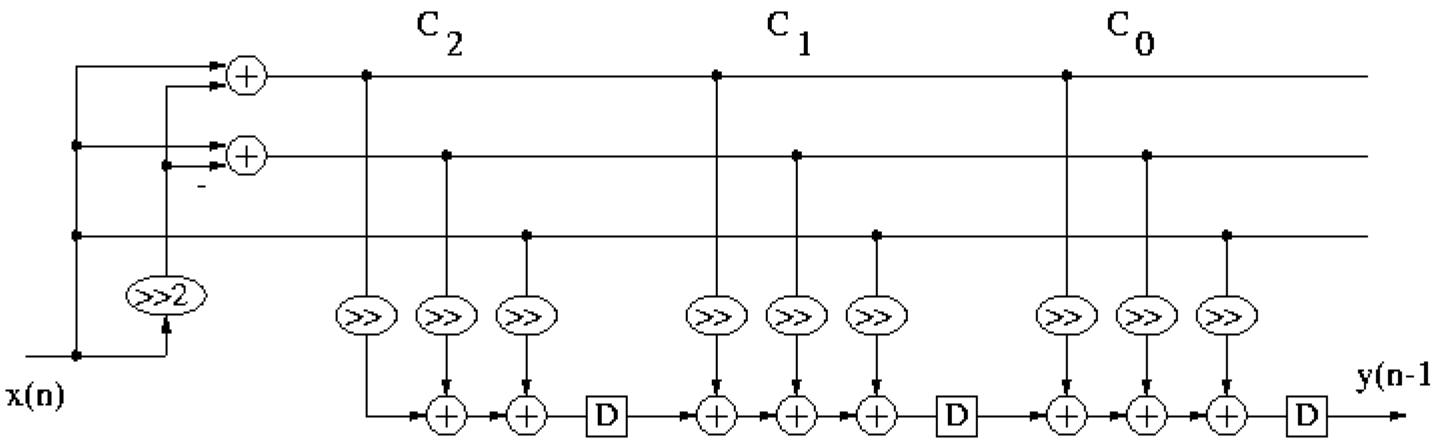
3-tap FIR filter with sub-expression sharing for 3-tap FIR filter with coefficients c_2 = 0.11010010, c_1 = 0.10011010 and c_0 = 0.00101011.
This requires 7 shifts and 9 additions compared to 12 shifts and 11 additions.



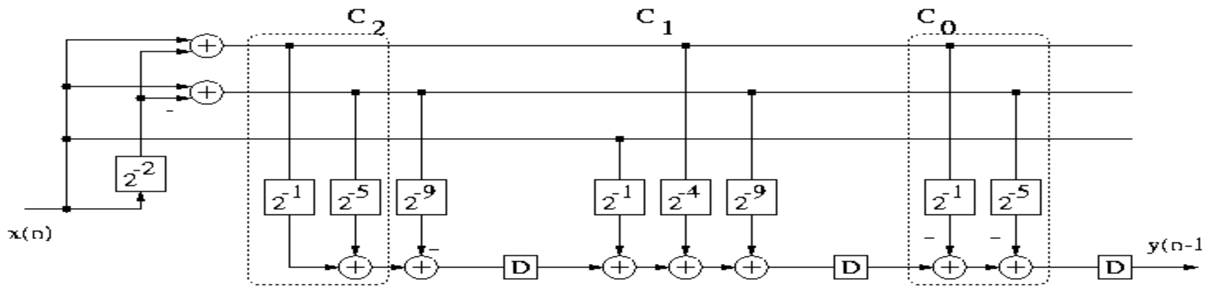
3-tap FIR filter with sub-expression sharing requiring 8 additions as compared to 9 in the previous implementation.

Using 2 most common sub-expressions in CSD representation

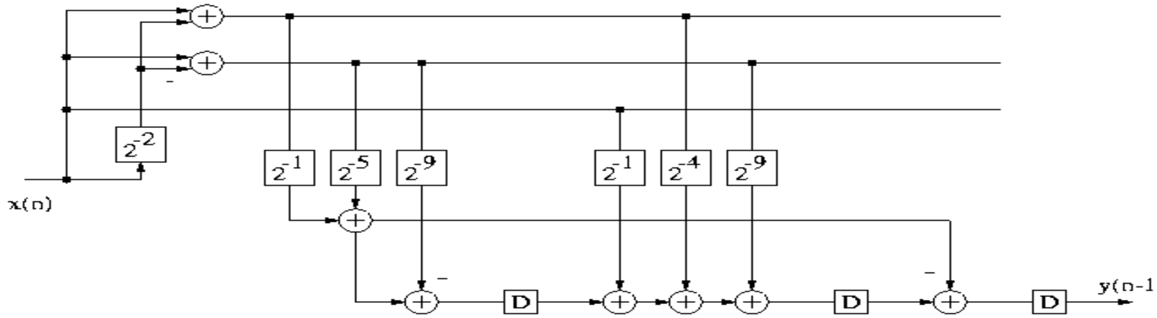
- x – x >> 2 and x + x >> 2 are the 2 most common sub-expressions in CSD representation.



An FIR filter using the term sharing, where the two most common sub-expressions in CSD numbers 101 and $10\bar{1}$, together with isolated 1 are shared among all filter coefficients.



(a)



(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.