TSTE87 ASIC for DSP
Lecture 11 2014

Oscar Gustafsson

May 20, 2014

Outline

Efficient polynomial multiplication

Introduction

- In 1960 Anatolii Alexeevitch Karatsuba attended a lecture by Andrey Kolmogorov at Moscow State University, where Kolmogorov presented his conjecture that a multiplication of two n-digit numbers requires $O(n^2)$ elementary operations
- The following week, Karatsuba presented an algorithm requiring $O(n^{\log_2 3})$ elementary operations
- Kolmogorov then wrote and published a paper with Karatsuba as the author without telling him

What was the idea?

- Consider a polynomial multiplication

$$(a_1x + a_0)(b_1x + b_0) = c_2x^2 + c_1x + c_0 \quad (1)$$

  where we want to compute c_2 to c_0
- A straightforward realization require the computation of

$$a_1b_1, a_0b_1, a_1b_0, a_0b_0 \quad (2)$$

  i.e., four multiplications
- Instead, by computing (e.g.) $(a_1 + a_0)(b_1 + b_0)$, a_1b_1, and a_0b_0 it is enough to perform three multiplications

$$c_2 = a_1b_1 \quad (3)$$
$$c_0 = a_0b_0 \quad (4)$$
$$c_1 = (a_1 + a_0)(b_1 + b_0) - c_0 - c_2 \quad (5)$$

But two term polynomials are not so common!?

- Wrong!
- Fixed-point multiplication using B-bit numbers (use polynomial variable $2^{B/2}$)

$$(a_1 2^{B/2} + a_0)(b_1 2^{B/2} + b_0) = c_2 2^B + c_1 2^{B/2} + c_0 \tag{6}$$

a_1, b_1 are the most significant parts, a_0, b_0 are the least significant parts
- Complex multiplication using polynomial variable j

$$(a_1 j + a_0)(b_1 j + b_0) = c_2 j^2 + c_1 j + c_0 = c_0 - c_2 + jc_1 \tag{7}$$

- FIR filters using polynomial variable z

$$(H_1(z^2)z + H_0(z^2)(X_1(z^2)z + X_0(z^2)) = c_2 z^2 + c_1 z + c_0 \tag{8}$$

where H_1, X_1 are the odd indexed values of impulse response and input data, respectively, and H_0, X_0 are the even indexed values

OK, but how can you determine these?

- Let us start with the property that an N-term polynomial is uniquely defined by the value of N points
- In addition, instead of using the polynomial itself one can use any derivative degree of it
- Let us consider the two-term polynomial multiplication

$$(a_1 x + a_0)(b_1 x + b_0) = c_2 x^2 + c_1 x + c_0 \tag{9}$$

- Now, evaluate the polynomial for the points $x = 0, 1, \infty$

$$x = 0 \;\Rightarrow\; a_0 b_0 = c_0 \tag{10}$$
$$x = 1 \;\Rightarrow\; (a_1 + a_0)(b_1 + b_0) = c_2 + c_1 + c_0 \tag{11}$$
$$x = \infty \;\Rightarrow\; a_1 b_1 = c_2 \tag{12}$$

Oh, I see...

- On matrix form this becomes

$$\begin{bmatrix} a_0 b_0 \\ (a_1 + a_0)(b_1 + b_0) \\ a_1 b_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} \tag{13}$$

- Now, to determine c_0 to c_2 invert the matrix as

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_0 b_0 \\ (a_1 + a_0)(b_1 + b_0) \\ a_1 b_1 \end{bmatrix} \tag{14}$$

- This can be further rewritten as

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_0 & 0 & 0 \\ 0 & a_1 + a_0 & 0 \\ 0 & 0 & a_1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} \tag{15}$$

Let us play around for a bit

- What other equations can one use?
- Evaluate for $x = 2$ instead of $x = 1$

$$(2a_1 + a_0)(2b_1 + b_0) = 4c_2 + 2c_1 + c_0 \tag{16}$$

$$\begin{bmatrix} a_0 b_0 \\ (2a_1 + a_0)(2b_1 + b_0) \\ a_1 b_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 2 & 4 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} \tag{17}$$

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ -\frac{1}{2} & \frac{1}{2} & -2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_0 & 0 & 0 \\ 0 & 2a_1 + a_0 & 0 \\ 0 & 0 & a_1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 2 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} \tag{18}$$

- Quite OK assuming shifts are free, but not better in any way

More alternatives

- Let us take a look at some alternatives

Point	Matrix row	Multiplication(s)
$x = 0$	$[1\ 0\ 0]$	$a_0 b_0$
$x = 1$	$[1\ 1\ 1]$	$(a_1 + a_0)(b_1 + b_0)$
$x = -1$	$[1\ -1\ 1]$	$(-a_1 + a_0)(-b_1 + b_0)$
$x = \infty$	$[0\ 0\ 1]$	$a_1 b_1$
$x = 2$	$[1\ 2\ 4]$	$(2a_1 + a_0)(2b_1 + b_0)$
$\partial x = 0$	$[0\ 1\ 0]$	$a_0 b_1 + a_1 b_0$
$\partial x = 1$	$[0\ 1\ 2]$	$(a_1 + a_0)b_1 + a_1(b_0 + b_1)$

- Which ones should we choose?
  - Matrix should be invertible
  - Matrix inverse should include "nice" values
  - Preferably only one multiplication per row

FIR filter aspects

- For FIR filters, the polynomial weights are the polyphase components
- Hence, if a symmetric FIR filter is used, the resulting polyphase components and sums of polyphase components may also be symmetric (or not)
- For FIR filters it is therefore of interest to find structures using as many symmetric sub filters as possible

FIR filter aspects

- Consider an even N-order (odd-length) linear-phase FIR filter
- Direct realization require $\frac{N}{2} + 1 \approx \frac{N}{2}$ multiplications per sample
- Polyphase components H_0 and H_1 are symmetric, $H_0 + H_1$ is not
- Leading to $\frac{N/4+N/4+N/2}{2} = \frac{N}{2}$ multiplications per sample
- For odd N, $H_0 + H_1$ is symmetric while H_0 and H_1 are not
- Leading to $\frac{N/2+N/2+N/4}{2} = \frac{5N}{8}$ multiplications per sample
- Increase compared to direct realization

FIR filter aspects

- Instead select to use $x = -1, 0, 1$ leading to

$$\begin{bmatrix} a_0 b_0 \\ (a_1 + a_0)(b_1 + b_0) \\ (a_0 - a_1)(b_0 - b_1) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & -1 & 1 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} \quad (19)$$

- Now, to determine c_0 to c_2 invert the matrix as

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{2} & -\frac{1}{2} \\ -1 & \frac{1}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} a_0 b_0 \\ (a_1 + a_0)(b_1 + b_0) \\ (a_0 - a_1)(b_0 - b_1) \end{bmatrix} \quad (20)$$

- This can be further rewritten as

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \\ -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} a_0 & 0 & 0 \\ 0 & \frac{a_1+a_0}{2} & 0 \\ 0 & 0 & \frac{a_0-a_1}{2} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} \quad (21)$$

- Complexity now $\frac{N/2+N/4+N/4}{2} = \frac{N}{2}$ multiplications per sample

Higher order polynomials

- For two term polynomials there are a limited number of cases resulting in three multiplications and only ones in the inverse
- However, for higher order polynomials (more terms) it becomes more challenging/interesting
- First, note that two polynomials with a power of two terms, say 2^N require $2^{N\log_2 3} = 3^N$ multiplications by applying the scheme recursively instead of 4^N multiplications
- Second, note that multiplying two polynomials with A and B terms respectively generates a polynomial with $A+B-1$ terms
- Hence, $A+B-1$ equations and therefore multiplications are required (but there may be non-trivial constant multiplications involved)

Three term polynomials

- For three terms let us evaluate the polynomials at $x = 0, 1, -1, -2, \infty$ leading to

$$\begin{bmatrix} a_0 b_0 \\ (a_2+a_1+a_0)(b_2+b_1+b_0) \\ (a_2-a_1+a_0)(b_2-b_1+b_0) \\ (4a_2-2a_1+a_0)(4b_2-2b_1+b_0) \\ a_2 b_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 \\ 1 & -2 & 4 & -8 & 16 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} \tag{22}$$

- The inverse of the matrix is now

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1/2 & 1/3 & -1 & 1/6 & -2 \\ -1 & 1/2 & 1/2 & 0 & -1 \\ -1/2 & 1/6 & 1/2 & -1/6 & 2 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{23}$$

Three term polynomials

- This matrix is non-trivial to implement since it includes fractions of three and six
- One simple way is to just multiply the matrix by six leading to exact results with a scaling error of six (three as one can shift)
- However, to avoid this let us consider the derivative of the polynomial multiplication

$$(a_2x^2+a_1x+a_0)(b_2x^2+b_1x+b_0) = c_4x^4+c_3x^3+c_2x^2+c_1x+c_0 \tag{24}$$

- The derivative of this is

$$\begin{aligned} &(2a_2x+a_1)(b_2x^2+b_1x+b_0)+ \\ &(a_2x^2+a_1x+a_0)(2b_2x+b_1) = \\ &\quad 4c_4x^3+3c_3x^2+2c_2x+c_1 \end{aligned} \tag{25}$$

- Evaluate at $x = 0$ leads

$$a_1 b_0 + a_0 b_1 = c_1 \tag{26}$$

Three term polynomials

- Leading to

$$\begin{bmatrix} a_0 b_0 \\ (a_2+a_1+a_0)(b_2+b_1+b_0) \\ (a_2-a_1+a_0)(b_2-b_1+b_0) \\ a_1 b_0 + a_0 b_1 \\ a_2 b_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} \tag{27}$$

- With the inverse

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ -1 & \frac{1}{2} & \frac{1}{2} & 0 & -1 \\ 0 & \frac{1}{2} & -\frac{1}{2} & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_0 b_0 \\ (a_2+a_1+a_0)(b_2+b_1+b_0) \\ (a_2-a_1+a_0)(b_2-b_1+b_0) \\ a_1 b_0 + a_0 b_1 \\ a_2 b_2 \end{bmatrix} \tag{28}$$

Three term polynomials

- However, as $a_1 b_0 + a_0 b_1$ can not be realized using a single filter one can write

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ -1 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & -1 \\ 0 & \frac{1}{2} & -\frac{1}{2} & -1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_0 b_0 \\ (a_2 + a_1 + a_0)(b_2 + b_1 + b_0) \\ (a_2 - a_1 + a_0)(b_2 - b_1 + b_0) \\ a_1 b_0 \\ a_0 b_1 \\ a_2 b_2 \end{bmatrix} \tag{29}$$

- Instead one can use more multiplications and end up with matrices only containing powers of two as e.g.

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ -1 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & -1 \\ 0 & \frac{1}{2} & -\frac{1}{2} & -1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \operatorname{diag} \begin{bmatrix} a_0 \\ a_0 + a_1 + a_2 \\ a_0 - a_1 + a_2 \\ a_1 \\ a_0 \\ a_2 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix} \tag{30}$$

Winograd's inner product algorithm

- For the previously discussed algorithms at least two samples are needed to iterate once
- In 1968 Winograd proposed the following way of computing an inner product

$$y_n = \underbrace{\sum_{k=0}^{\frac{N+1}{2}-1} (x_{n-(2k+1)} + h_{2k})(x_{n-2k} + h_{2k+1})}_{b_n} - \underbrace{\sum_{k=0}^{\frac{N+1}{2}-1} h_{2k} h_{2k+1}}_{c} - \underbrace{\sum_{k=0}^{\frac{N+1}{2}-1} x_{n-2k} x_{n-(2k+1)}}_{d_n} . \tag{31}$$

- Let us consider the different terms from an FIR filter perspective

Winograd-based FIR filters

- The c term is just a constant for a given FIR filter

$$c = \sum_{k=0}^{\frac{N+1}{2}-1} h_{2k} h_{2k+1} \tag{32}$$

- The d_n term can be computed recursively requiring only a single multiplication per iteration

$$d_n = \sum_{k=0}^{\frac{N+1}{2}-1} x_{n-2k} x_{n-(2k+1)} \tag{33}$$

Note that two different results for even and odd n must be stored, also for one multiplication per iteration the product must be stored

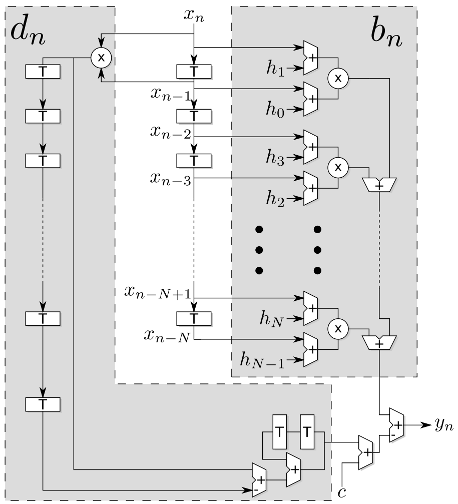
Winograd-based FIR filters

- Finally, the b_n term require $\frac{N+1}{2}$ muliplications per iteration

$$b_n = \sum_{k=0}^{\frac{N+1}{2}-1} (x_{n-(2k+1)} + h_{2k})(x_{n-2k} + h_{2k+1}) \tag{34}$$

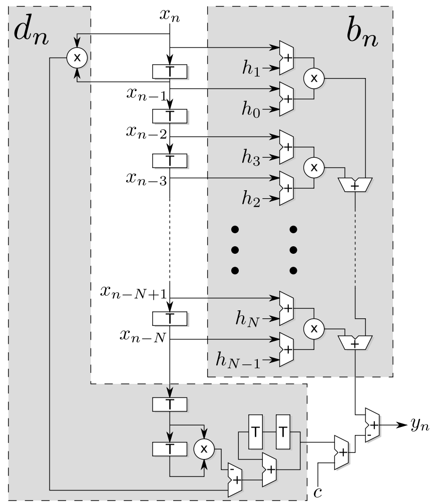
Winograd-based FIR filters

- The resulting FIR filter using one multiplication per d_n iteration looks like



Winograd-based FIR filters

- The resulting FIR filter using two multiplications per d_n iteration looks like



Complexity per sample comparison

Odd-order FIR filters not considering possible symmetry

Architecture	Mult	Add/Sub	Delays
Direct form	$N+1$	N	N
Poly-phase two-parallel	$N+1$	N	$\frac{N}{2}$
Polynomial two-parallel	$\frac{3(N+1)}{4}$	$\frac{3N+5}{4}$	$\frac{3N-1}{4}$
Polynomial two-parallel (fewer reg.)	$\frac{3(N+1)}{4}$	$N+1$	$\frac{N}{2}$
Winograd sequential (two mult per d_n)	$\frac{N+1}{2}+2$	$\frac{3(N+3)}{2}$	$N+4$
Winograd sequential (one mult per d_n)	$\frac{N+1}{2}+1$	$\frac{3(N+3)}{2}$	$2N+2$
Winograd two-parallel (two mult per d_n)	$\frac{N+1}{2}+2$	$\frac{3(N+3)}{2}$	$\frac{N+7}{2}$
Winograd two-parallel (one mult per d_n)	$\frac{N+1}{2}+1$	$\frac{3(N+3)}{2}$	$N+1$

Squaring-based multiplication

$$(a+b)^2 = a^2 + b^2 + 2ab \Rightarrow ab = \frac{(a+b)^2 - a^2 - b^2}{2} \qquad (35)$$

- Can be efficient when implementing multiplication using lookup tables
- Can also be used for FIR filters, complex multipliers, matrix multiplication etc