Application Specific Integrated Circuits for Digital Signal Processing
Lecture 9

Oscar Gustafsson

Today's topics

- Number representations
- Addition

(If you enjoy this lecture and the next,
take the TSTE18 Digital Arithmetic course)

Number representations

- The number representation defines both the arithmetic operations and numerical properties
- Fixed-point arithmetic has been dominating for a long time because of the less complex operations
- Floating-point arithmetic is gaining ground, especially for numerically troublesome applications such as MIMO detection
- Non-traditional number representations such as logarithmic number systems and residue number systems are advantageous in certain applications (more on next lecture)
- We will primarily focus on fractional numbers, but most of the time this is just a matter of indexes

Number representations

- A W fractional digits radix-r positive fractional number, $0 \leq X < 1$, is written as

$$X = \sum_{i=1}^{W} x_i r^{-i} \tag{1}$$

  where $x_i \in \{0, 1, \ldots, r-1\}$
- For most cases we assume $r = 2$, i. e., a binary representation, which gives

$$X = \sum_{i=1}^{W} x_i 2^{-i} \tag{2}$$

  where $x_i \in \{0, 1\}$ are binary digits, i. e., bits
- Define the weight of the least significant bit as $Q = 2^{-W}$

Binary addition

- Add the two binary numbers

$$X = \sum_{i=1}^{W} x_i 2^{-i} \text{ and } Y = \sum_{i=1}^{W} y_i 2^{-i} \tag{3}$$

- The result can be written as

$$S = c_0 + \sum_{i=1}^{W} s_i 2^{-i} \tag{4}$$

where

$$s_i = x_i \oplus y_i \oplus c_i$$

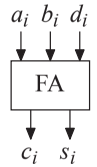
and

$$c_{i-1} = \text{majority}(x_i, y_i, c_i) = x_i y_i + y_i c_i + x_i c_i$$

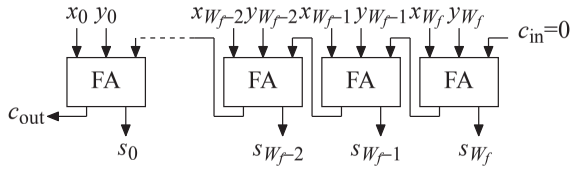
with $c_W = 0$

Binary addition

- These equations are often realized with a *full adder* cell employing three inputs, x_i, y_i, and c_i, and two outputs, s_i and c_{i-1}



- Several full adders are connected to form a ripple carry adder



- Gate delay proportional to the number of bits

Negative numbers

- Easy to represent positive numbers, but what about negative?
- Representing negative numbers are required for typical DSP applications, although sometimes one can just offset the signals
- Subtraction can be written and computed as $X - Y = X + (-Y)$
- To illustrate the different number representations we will consider the following properties and operations
  - Representation and numerical range
  - Increasing the number of bits on the least significant side (left shift) and on the most significant side (right shift/sign-extension)
  - Negation
  - Addition/subtraction
  - Multiplication
- Assume one more bit compared to before to represent the sign

Sign-magnitude

- Use one bit to represent the sign and the rest of the bits for the magnitude

$$X = (1 - 2x_0) \sum_{i=1}^{W} x_i 2^{-i} = (-1)^{x_0} \sum_{i=1}^{W} x_i 2^{-i} \tag{5}$$

- Numerical range $-1 + Q \leq X \leq 1 - Q$
- Examples
  - $0 = 0.000$ or 1.000
  - $\frac{5}{8} = 0.101$
  - $-\frac{5}{8} = 1.101$
- Increasing wordlength on MSB side (sign extend)/right shift
  - Add zeros after sign bit
  - $\frac{5}{8} = 00.101$
  - $-\frac{5}{8} = 10.101$
  - $\frac{5}{16} = 0.0101$
  - $-\frac{5}{16} = 1.0101$

Sign-magnitude

- Increasing wordlength on LSB side/left shift
  - Add zeros after the LSB
  - $\frac{5}{8} = 0.1010$
  - $-\frac{5}{8} = 1.1010$
  - $\frac{5}{4} = 01.010$
  - $-\frac{5}{4} = 11.010$
- Negation
  - Invert sign-bit
- Addition/subtraction
  - Complicated, must determine which one is the largest in relation to the sign and operation to be performed
- Multiplication
  - Unsigned multiplication of magnitudes, xor of sign-bits

One's complement

- Negate by inverting all bits

$$X = -x_0(1 - 2^{-W}) + \sum_{i=1}^{W} x_i 2^{-i} \tag{6}$$

- Numerical range $-1 + Q \leq X \leq 1 - Q$
- Examples
  - $0 = 0.000$ or 1.111
  - $\frac{5}{8} = 0.101$
  - $-\frac{5}{8} = 1.010$
- Increasing wordlength on MSB side (sign extend)/right shift
  - Add sign-bit before the MSB
  - $\frac{5}{8} = 00.101$
  - $-\frac{5}{8} = 11.010$
  - $\frac{5}{16} = 0.0101$
  - $-\frac{5}{16} = 1.1010$

One's complement

- Increasing wordlength on LSB side/left shift
  - Add sign-bit after the LSB
  - $\frac{5}{8} = 0.1010$
  - $-\frac{5}{8} = 1.0101$
  - $\frac{5}{4} = 01.010$
  - $-\frac{5}{4} = 10.101$
- Negation
  - Invert all bits
- Addition/subtraction
  - Perform normal binary addition, then add the carry output c_{-1} to the carry input c_W
  - End-around carry
  - Example:

$$\underbrace{0.10}_{\frac{1}{2}} + \underbrace{1.10}_{-\frac{1}{4}} = \underbrace{1}_{c_{-1}}\,0.00 \Rightarrow 0.00 + \underbrace{0.01}_{c_W = c_{-1}} = \underbrace{0.01}_{\frac{1}{4}} \tag{7}$$

- Multiplication
  - Complicated handling of sign-bits on both LSB and MSB side

Two's complement

$$X = -x_0 + \sum_{i=1}^{W} x_i 2^{-i} \tag{8}$$

- Numerical range $-1 \leq X \leq 1 - Q$
- Examples
  - $0 = 0.000$
  - $\frac{5}{8} = 0.101$
  - $-\frac{5}{8} = 1.011$
- Increasing wordlength on MSB side (sign extend)/right shift
  - Add sign-bit before the MSB
  - $\frac{5}{8} = 00.101$
  - $-\frac{5}{8} = 11.011$
  - $\frac{5}{16} = 0.0101$
  - $-\frac{5}{16} = 1.1011$

Two's complement

- Increasing wordlength on LSB side/left shift
  - Add zeros after the LSB
  - $\frac{5}{8} = 0.1010$
  - $-\frac{5}{8} = 1.0110$
  - $\frac{5}{4} = 01.010$
  - $-\frac{5}{4} = 10.110$
- Negation
  - Invert all bits and add one at the LSB position
- Addition/subtraction
  - Normal binary addition
  - Example:

$$\underbrace{0.10}_{\frac{1}{2}} + \underbrace{1.11}_{-\frac{1}{4}} = \cancel{1}\underbrace{0.01}_{\frac{1}{4}} \tag{9}$$

  - For subtraction there is no need to evaluate the negation explicitly, instead the additional one can be added at the c_W input of the adder
- Multiplication
  - Complicated handling of sign-bits on MSB side

Number representations

- Two's complement is the commonly used representation
  - Also has the benefit of adding numbers in arbitrary order as discussed earlier
- Two's complement has one "major" issue to consider
  - Multiplying -1 with -1 leads to 1 which is out of range
  - Will have to decide if an additional integer bit should be added for this specific case or if one detects it and use saturation to $1 - Q$
- Sign-magnitude has benefits in terms of switching activity
  - Consider a value changing from Q to 0 to $-Q$
    - Sign-magnitude:
      $0.000\ldots001 \rightarrow 0.000\ldots000 \rightarrow 1.000\ldots001$
    - Two's complement:
      $0.000\ldots001 \rightarrow 0.000\ldots000 \rightarrow 1.111\ldots111$
  - This is the typical case for many "silent" signals, only some noise
  - Can be utilized e.g. by having one accumulator for negative numbers and one for positive

Signed-digit (SD) representation

- Allow each "bit" to be signed, i.e., $x_i \in \{-1, 0, 1\}$

$$X = \sum_{i=1}^{W} x_i r^{-i} \tag{10}$$

- Numerical range $-1 + Q \leq X \leq 1 - Q$
- The "bit" is a ternary digit, sometimes called trit
- Require two bits to represent but there are other advantages
- Several representations of many numbers (denote -1 as $\bar{1}$)
  - Example: $\frac{5}{8} = 0.101 = 1.\bar{1}01 = 1.\bar{1}1\bar{1} = 1.0\bar{1}\bar{1}$
  - Redundant number system
- Main benefit: addition/subtraction can be made without carry-propagation

Signed-digit number systems

- Conversion from two's complement to signed-digit
  - Only the sign-bit must be considered
  - Examples:
    - $\frac{5}{8} = 0.101_{2C} = 0.101_{SD}$
    - $-\frac{5}{8} = 1.011_{2C} = \bar{1}.011_{SD}$
- Conversion from signed-digit to two's complement
  - Subtract the number formed by the negative digits from that formed by the positive digits
  - Example:
    - $\frac{5}{8} = 1.\bar{1}1\bar{1}_{SD} = 1.010_2 - 0.101_2 = 1.010 + \underbrace{1.010 + 0.001}_{\text{2C negation}} =$

      $\cancel{1}0.101_{2C}$

Minimum signed-digit representation

- A signed-digit representation with minimum number of non-zero digits is called a minimum signed-digit (MSD) representation
- In general there are several MSD representations
- One particular MSD representation can be obtained if we constraint two adjacent digits to not both be non-zero

$$x_i x_{i+1} = 0 \tag{11}$$

- This particular representation is called canonic signed digit (CSD) representation
- The CSD representation is unique and therefore not redundant
- This is useful as a SD representation without adjacent non-zero digits are then known to be minimum

Minimum signed-digit representation

- Properties of CSD vs binary

	Binary	CSD
Average non-zero digits	$W/2$	$W/3$
Maximum non-zero digits	W	$W/2$

- Conversion to CSD
  - Convert strings of ones: $011\ldots11 \Rightarrow 100\ldots0\bar{1}$
  - Merge adjacent digits of opposite signs: $1\bar{1} \Rightarrow 01$, $\bar{1}1 \Rightarrow 0\bar{1}$

Carry-save representation

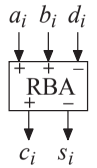
- Consider a ripple carry adder where the carries are not connected to the next stage
- Three words can be added and the result is represented using two outputs (carry and sum)
- Assuming two's complement representation

$$X = S + C = -s_0 + \sum_{i=1}^{W} s_i 2^{-i} - c_0 + \sum_{i=1}^{W} c_i 2^{-i} \tag{12}$$

- Suitable for high-speed realizations as no carry is propagated
- To detect sign (i.e. hard to compare and detect overflows) a carry propagation is required
- Can be seen as a signed-digit representation with $x_i \in \{0, 1, 2, 3\}$ and $x_i = 2c_i + s_i$

Redundant binary representation

- An alternative sometimes usable is the redundant binary representation
- Generalization of carry-save by writing $x_i = 2c_i + s_i$ as $x_i = 2c_i - s_i$ or $x_i = -2c_i + s_i$
- The first equation can represent $x_i \in \{-1, 0, 1, 2\}$ so it can add one negative and two positive inputs, while the other can represent $x_i \in \{-2, -1, 0, 1\}$ and add two negative and one positive
- Compare with a radix-2 signed digit, which can represent $x_i \in \{-1, 0, 1\}$, still two bits are needed for the three values
- Redundant binary adder with one negative and two positive inputs



Addition

- Ripple carry adders are the base line adders
- Gate delay linear in the word length
- Example

Value		0	1	2	3	4	5	6	7	Signal
$-53/256$		1	1	0	0	1	0	1	1	x_i
$94/256$		0	1	0	1	1	1	1	0	y_i
	1	1	0	1	1	1	1	0		c_i
$41/256$		0	0	1	0	1	0	0	1	s_i

- Becomes problematic for high-speed implementations
- Several methods to speed addition up will be covered
- Focus on parallel prefix addition

Carry-look ahead addition

- The carry can either take on a value independent on the carry input or the exact value of the carry input
- These cases are

a_i	b_i	c_{i-1}	Case
0	0	0	No carry-propagation (kill)
0	1	c_i	Carry-propagation (propagate)
1	0	c_i	Carry-propagation (propagate)
1	1	1	Carry-generation (generate)

- We can define propagate, p_i, and generate, g_i at bit-level as

$$p_i = a_i \oplus b_i \text{ and } g_i = a_i b_i \tag{13}$$

Carry-look ahead addition

- The carry output can then be expressed as

$$c_{i-1} = g_i + p_i c_i \tag{14}$$

- For the next stage the expression becomes

$$\begin{aligned} c_{i-2} &= g_{i-1} + p_{i-1}c_{i-1} = g_{i-1} + p_{i-1}(g_i + p_i c_i) \\ &= g_{i-1} + p_{i-1}g_i + p_{i-1}p_i c_i \end{aligned} \tag{15}$$

- For $N+1$:th stage we have

$$\begin{aligned} c_{i-(N+1)} &= g_{i-N} + p_{i-N}g_{i-(N-1)} + p_{i-N}p_{i-N-1}g_{i-(N-2)} + \\ &\quad + \cdots + p_{i-(N-1)} \cdots p_{i-1}p_i c_i \end{aligned} \tag{16}$$

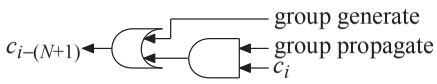
Carry-look ahead addition

- These terms can be grouped as

$$\begin{aligned} c_{i-(N+1)} &= \underbrace{g_{i-N} + p_{i-N}g_{i-(N-1)} + p_{i-N}p_{i-N-1}g_{i-(N-2)} + \cdots +}_{G_{i-N:i}} \\ &\quad + \underbrace{p_{i-(N-1)} \cdots p_{i-1}p_i}_{P_{i-N:i}} c_i \\ &= G_{i-N:i} + P_{i-N:i}c_i \end{aligned}$$

where $G_{i-N:i}$ is called group generate and $P_{i-N:i}$ group propagate as they correspond to the generate and propagate signals for the group of bits $i-N$ to i

- The group generate and propagate signals can be computed before the incoming carry is known to speed up the output carry generation



Parallel prefix addition

- The carry look-ahead concept can be formalized by introducing the dot operator

$$\begin{bmatrix} g_k \\ p_k \end{bmatrix} = \begin{bmatrix} g_i \\ p_i \end{bmatrix} \bullet \begin{bmatrix} g_j \\ p_j \end{bmatrix} \triangleq \begin{bmatrix} g_i + p_i g_j \\ p_i p_j \end{bmatrix} \tag{17}$$

- Adders based on dot operators are commonly referred to as parallel prefix adders
- The group generate and propagate for bits k to l can be written as

$$\begin{bmatrix} G_{k:l} \\ P_{k:l} \end{bmatrix} \triangleq \begin{bmatrix} g_k \\ p_k \end{bmatrix} \bullet \begin{bmatrix} g_{k+1} \\ p_{k+1} \end{bmatrix} \bullet \cdots \bullet \begin{bmatrix} g_l \\ p_l \end{bmatrix} \tag{18}$$

Parallel prefix addition

- The dot-operator is associative

$$\begin{bmatrix} g_k \\ p_k \end{bmatrix} = \left(\begin{bmatrix} g_i \\ p_i \end{bmatrix} \bullet \begin{bmatrix} g_j \\ p_j \end{bmatrix} \right) \bullet \begin{bmatrix} g_l \\ p_l \end{bmatrix} = \begin{bmatrix} g_i \\ p_i \end{bmatrix} \bullet \left(\begin{bmatrix} g_j \\ p_j \end{bmatrix} \bullet \begin{bmatrix} g_l \\ p_l \end{bmatrix} \right)$$

- Idempotent

$$\begin{bmatrix} g_k \\ p_k \end{bmatrix} = \begin{bmatrix} g_k \\ p_k \end{bmatrix} \bullet \begin{bmatrix} g_k \\ p_k \end{bmatrix}$$

- But not commutative

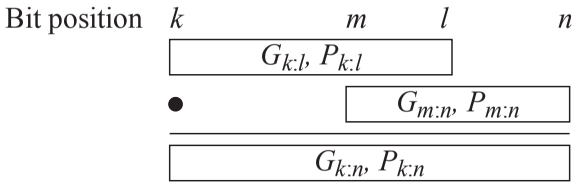
$$\begin{bmatrix} g_k \\ p_k \end{bmatrix} = \begin{bmatrix} g_i \\ p_i \end{bmatrix} \bullet \begin{bmatrix} g_j \\ p_j \end{bmatrix} \neq \begin{bmatrix} g_j \\ p_j \end{bmatrix} \bullet \begin{bmatrix} g_i \\ p_i \end{bmatrix}$$

Parallel prefix addition

- The idempotency leads to the very useful property that overlapping groups do not cause any problems

$$\begin{bmatrix} G_{k:n} \\ P_{k:n} \end{bmatrix} = \begin{bmatrix} G_{k:l} \\ P_{k:l} \end{bmatrix} \bullet \begin{bmatrix} G_{m:n} \\ P_{m:n} \end{bmatrix}, \; k \leq l, m \leq n, m \leq l-1. \tag{19}$$

which can be illustrated as below



Parallel prefix addition

- The carry signal in position k can be written as

$$c_k = G_{(k+1):l} + P_{(k+1):l} c_l \tag{20}$$

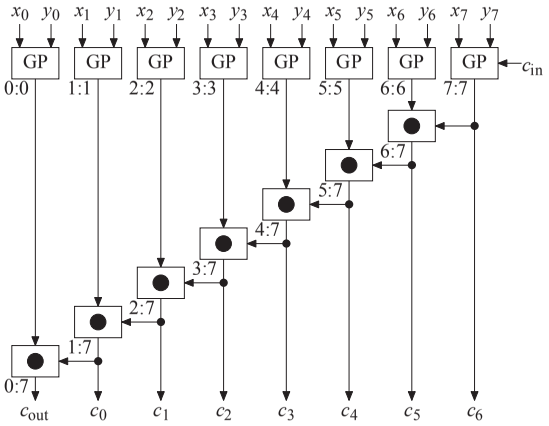
- Similarly, the sum signal in position k for an adder using W fractional bits is then expressed as

$$s_k = a_k \oplus b_k \oplus d_k = p_k \oplus (G_{(k+1):W} + P_{(k+1):W} c_{\text{in}}) = p_k \oplus c_k \tag{21}$$

- Of interest to compute all group generate and group propagate originating from the LSB position, i.e., $G_{k:W}$ and $P_{k:W}$ for $1 \leq k \leq W$.

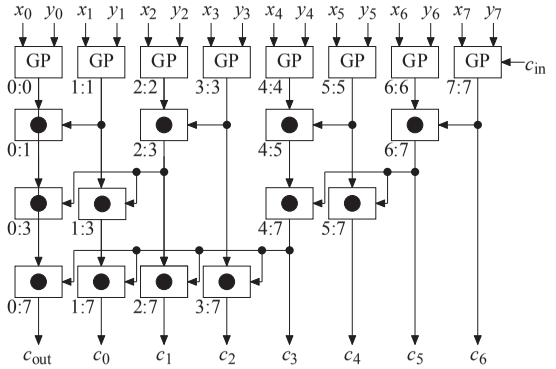
Parallel prefix addition

- Can be computed sequentially



Parallel prefix addition

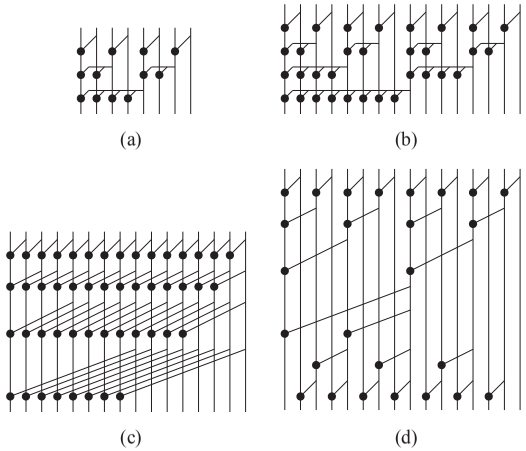
- But we can also use the properties of the dot product and compute it in parallel



- Many different structures for computing the dot product trees have been proposed trading dot operators, fan-out, wire length, etc.

Parallel prefix addition

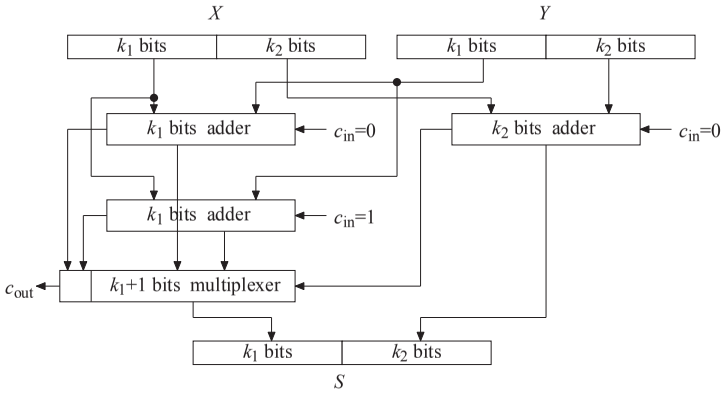
- Often a simplified view is used to represent the dot-products



- (a) 8-bit Ladner-Fisher (compare previous slide), (b) 16-bit Ladner-Fisher, (c) 16-bit Kogge-Stone, and (d) 16-bit Brent-Kung

Carry-select and conditional sum addition

- At any stage of the adder, the carry signal is either zero or one(!)
- Precompute both results and select the correct result once the correct carry arrives
- Carry-select adder



Carry-select and conditional sum addition

- The carry-select adder can of course be separate into several stages
- Best selection depends on the gate delays of the adders and the multiplexers
- Each pre-computation adder can in turn be split into a carry-select adder
- If this is done until the word length of each adder is one a conditional sum adder is obtained

Summary

- Binary number representations
  - Main problem is how to represent negative numbers
  - Different approaches have different pros and cons
  - Two's complement mainly used
- Redundant number representations
  - Redundant = several possible representations of many numbers
  - Can avoid carry propagation
  - Must eventually converted back to non-redundant form which require a carry propagation
  - Carry-save is easy to use since it relates very much to two's complement
- Addition
  - Main problem to mitigate is the lengthy carry propagation
  - Carry look-ahead/parallel prefix using dot operators
  - Can reorder and speed up the evaluation at the cost of more dot operators
  - Carry-select is an intuitive speed giving large speed up at reasonable cost