11.1 Assume that the input data are pre-scaled so that no overflow exits, assume also that the bit-parallel adder are two ripple-carry-adders.

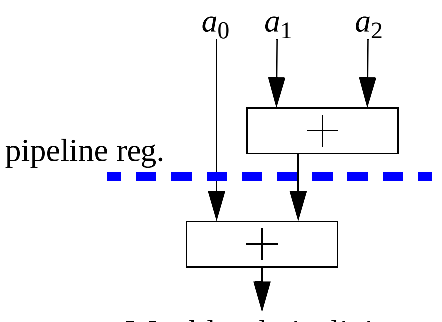
(a) The addition time for adder 1 and 2 is $(N-1)t_{carry} + t_{sum}$. Note that the addition for adder 2 can start as the LSB in the sum of a_1 and a_2 is available, i.e., the delay time is only t_{sum}. The total addition time is

$((N-1)t_{carry} + t_{sum}) + t_{sum} = 19t_{carry} + 2t_{sum} = 42$ ns

(b) The pipelining can be inserted at both word-level and bit-level.

The latency is two clock periods, where one clock period must be larger than $(N-1)t_{carry} + t_{sum}$, or 40 ns.

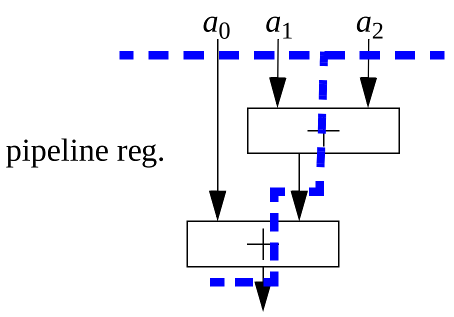
For the word-level, the new throughput is $\dfrac{1}{(N-1)t_{carry} + t_{sum}} = \dfrac{1}{40 \times 10^{-9}}$ (sample/s), or 25 Msample/s.



Word-level pipelining

The bit-level pipelining is more efficient.

Set the pipeline register at the $\left\lceil \frac{N}{2} \right\rceil$ th full adder of the first adder and the $\left\lceil \frac{N}{2} + 1 \right\rceil$ th full adder of the second adder.



Bit-level pipelining

The addition time is

$$max\left\{\left(\left\lceil \frac{N}{2} - 1 \right\rceil\right)t_{carry} + 2t_{sum}, \left(N - \left\lceil \frac{N}{2} \right\rceil - 1\right)t_{carry} + t_{sum}\right\} = 22 \text{ ns.}$$

The new throughput is about 45 Msample/s.

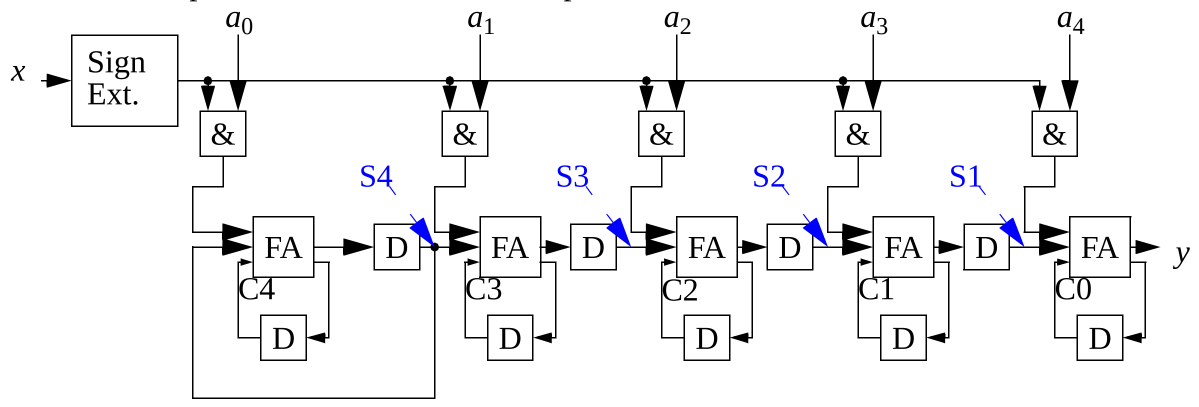
The latency in this case is to clock periods with clock period no less than 22ns.

11.2 Multiplication with Booth recoding for $15_{10} \times (-13)_{10}$.

A		11	00	11		-13	
X	$\times$	00	11	11		15	
Y		+A	-0	-A		Recoded multiplier operation	
Add -A	+	00	11	01			
Shift 2-bit		00	00	11	01		
Add -0	+	00	00	00			
		00	00	11	01		
Shift 2-bit		00	00	00	11	01	
Add +A	+	11	00	11			
		11	00	11	11	01	-195

11.3 a) $D = 13 \cdot 11 \cdot 9 \cdot 7 \cdot 5 \cdot 4 = 180180$ or $log_2(180180) = 17.46$ bits.
b) $D = m_1 \cdot m_2 \cdot ...m_7 = 8\ 485\ 840\ 800$ which is greater than 2^{32}.

11.4 (a) $a = 1{,}1101$ and $x = 1{,}001$. The sign extension circuit extends the coefficient x to $W_d + W_c - 1 = 5 + 4 - 1 = 8$ bits, or, $x_{ext} = 11111{,}001$.
The multiplication with bit-serial multiplier is shown below.



Serial/parallel multiplier

	x	S4	S3	S2	S1	C4	C3	C2	C1	C0	y	
reset	0	0	0	0	0	0	0	0	0	0	0	(LSB)
x_{-3}	1	1	1	1	0	0	0	0	0	0	1	
x_{-2}	0	1	1	1	1	0	0	0	0	0	0	

	x	S4	S3	S2	S1	C4	C3	C2	C1	C0	y	
x_{-1}	0	1	1	1	1	0	0	0	0	0	1	
x_0	1	0	0	0	1	1	1	1	0	1	0	
x_1	1	0	0	0	0	1	1	1	0	1	1	
x_2	1	0	0	0	0	1	1	1	0	1	0	
x_3	1	0	0	0	0	1	1	1	0	1	0	
x_4	1	0	0	0	0	1	1	1	0	1	0	(MSB)

The result is $0{,}00101010_2 = 0{,}1640625_{10}$.

$ax = (1{,}1101)_2 \cdot (1{,}001)_2 = (-0{,}1875)_{10} \cdot (-0{,}875)_{10} = (0{,}1640625)_{10}$.

(b) The S/P multiplier in figure 11.15 needs $(W_d + W_c - 1) + 1$ clock periods to process one data, so the throughput is $\frac{1}{(16 + 4 - 1) + 1} = \frac{1}{20}$ sample per clock period.

With S/P multiplier in Figure 11.45, the throughput is doubled, i.e., $\frac{1}{10}$ sample per clock period.

11.5 a) Block diagram for bit-serial multiplier with coefficient $\alpha = (0{,}0101)_2$ is shown as following

If the most significant bit in the coefficient is 0, then the product for this bit is always 0 and hence the result is always 0 or the output from this bit can be replaced with a 0. The resulting block diagram is shown bellow.

For the same reason, we can simplify the next significant bit and the output signal is 0. The next significant bit is 1. The output from AND-gate is therefore equal to the input signal x. Since the input from the preceding bit is zero and the carry D-flip-flop is reset to 0 at the beginning, the carry output from this bit is 0. The sum for the addition is x. Therefore this bit can be replaced with a D-flip-flop.

The coefficient for the next bit is 0, with the reason, the carry and the input is always 0. Therefore we can simplify this bit to a D-flip-flop.

The coefficient at the last significant bit is 1 and should add result from the previous bit. There is no simplification at this bit. The resulting multiplier is shown below.

b) Control signal: assume that the word length is W_d for the data and W_c for the coefficient. Before the computation, all carry D-flip-flops have to reset to 0. The multiplication have to take $W_d + W_c - 1$ clock cycles, where $W_c - 1$ clock cycles are needed for sign extension.

c) A new multiplication can started after
$W_d + W_c - 1 = 12 + 5 - 1 = 16$ clock cycles.
In some case, one extra clock cycle is required for the reset of all D-flip-flops. This depends on the ways of realization.

d) Verification

11.6 a) Blockdiagram. A CSDC number C can be written as subtraction between two binary numbers $(C_+)_2$ and $(C_-)_2$, where $(C_+)_2$ is the positive part of the CSDC number (replace the negative ones with zeros) and $(C_-)_2$ is the negative part of the CSDC number (replace the positive ones with zeros and the the negative ones with ones):
$(C)_{\mathrm{CSDC}} = (C_+)_2 - (C_-)_2$.

If the LSB in the binary number x has value of 2^{-n} and the coefficient C is a CSDC number, tthe product y can be expressed as follows

$$y = Cx = (C_+ - C_-)x = C_+x + C_-(-x) = C_+x + C_-(x' + 2^{-n}) = C_+x + C_-x' + C_-2^{-n}$$

where x' is the bit-wise inversion of x.
In this execise, $\alpha = (0{,}100\bar{1})_{\mathrm{CSDC}}$ and LSB has a value of 2^{-7}. The product can be computed as $y = C_+x + C_-x' + C_-2^{-n} = (0{,}1000)_2x + (0{,}0001)_2x' + (0{,}0001)_2 2^{-7}$.
The multiplication with $(0{,}1000)_2$ and $(0{,}0001)_2$ is only shift operations. Moreover, the shift operation is embedded in the serial/parallel multipliers. The block diagram is shown below.

Obviously this block diagram can be simplified and the simplified block diagram is shown below.

b) Verification with $x = (0{,}110)_2$.

11.7 a) Block diagram is shown below. The numer of 1s in the coefficient $\alpha = (1{,}11011)_2$ is large and the usual method for optimization does not give too much simplification. A more optimal simplification can be achieved by changing the sign of both data and coefficient. The changing of sign of a two's compliment number can be done by bitwise inversion and add 1 at the LSB.

$$y = \alpha x = (-\alpha)(-x) = (-\alpha)(\bar{x} + 2^{-n}) = (-\alpha)\bar{x} + (-\alpha)2^{-n}$$

where $\bar{x}$ is the bitwise inversion of x.
The optimized block diagram is shown below.

b) Verification

11.8 The simplest way to implement serial/parallel multiplier with fixed coefficient is to choose between different number representation. We give the representation for the simplest implementation here and the block diagrams are left to the readers.
(a) $(0{,}011001)_2$

(b) $(0{,}111011)_2 = -(1{,}0000101)_2$, inverse the signs of input data and coefficient(see problem 11.7).

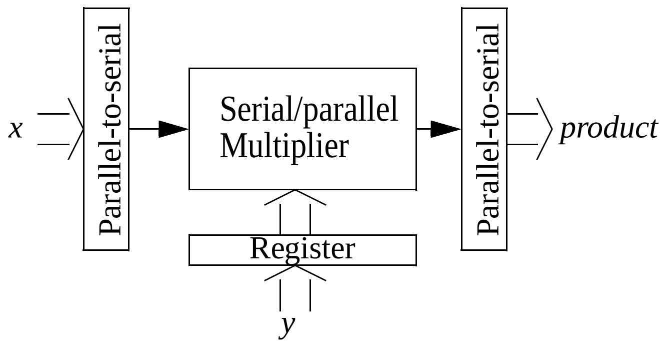
(c) $(1{,}011001)_2 = -(0{,}100111)_2 = -(0{,}10100\bar{1})_2$, inverse the signs of input data and coefficient(see problem 11.7) and use CSDC representation.

(d) $(1{,}011001)_2 = -(0{,}100101)_2$, inverse the signs of input data and coefficient.

(e) $(0{,}000001)_2$.

(f) $(1{,}000001)_2$.

11.11 The bit-serial PE can be realized with a parallel-to-serial converter and a serial-to-parallel converter and a serial/parallel multiplier (assume that the word length for x is larger than that of y). The block diagram is shown below.



Bit-serial PE

The serial/parallel multiplier can be the same as in Figure 11.15.

11.14 See Chapter 11 for distributed arithmetic.
The number range for the values stored in the ROM must be increased to [–2,þ2[. One shift-accumulator that can accumulate word with a word length of six bits, i.e., with six bit-slices, is required.

11.16 The number of inner products that shall be computed are 3, two for internal values and one for the output. Hence, 3 units are needed. The first unit has 3 inputs, x(n) and two branches in the recursive part, and 8 words. The second unit has 3 inputs from the first section and 2 branches from the recursive part, hence, 32 words. The last unit has only 3 inputs, 8 words.

11.17 a) The least significant bit in W_{ROM} corresponds to the least significant bit in any of the coefficients a_i. The most significant bit in W_{ROM} is determined by the most significant bit in the ROM. Hence, by the value with the largest magnitude that is stored in the ROM. This value is equal to: $max\ \{\sum a_{i+}, \sum a_{i-}\}$

That is the sum of all positive coefficients or the negative of the sum of all negative coefficients. The word length of the ROM, i.e., W_{ROM} determine the width of the shift-accumulator while W_d effect the execution time.

b) The throughput is inversely proportional to the largest value of W_d and W_{ROM}.

11.18 The set of difference equations in computable order is

$$u_2 := a_2\, x(n) + b_1\, v_3(n)$$
$$u_6 := a_3\, x(n) + a_4\, v_5(n) + b_2\, v_7(n) + b_3\, v_8(n)$$
$$y(n) := a_1\, x(n) + c_1 u_2 + c_2\, u_6$$
$$v_8(n) := v_7(n-1)$$
$$v_7(n) := u_6$$
$$v_3(n) := u_2$$
$$v_5(n) := x(n)$$

Hence, only three (Distributed arithmetic) units are needed. All units can work in parallel.

Unit	Inputs	Ouput	Number of terms	Number of words in the ROM
1	x(n), v_3(n)	u_2	2	4
2	x(n), v_5(n), v_7(n), v_8(n)	u_6	4	16
3	x(n), u_2, u_6	y(n)	3	8

11.22 A multiplication between the two complex numbers $a + j\,b$ and $c + j\,d$ can be rewritten

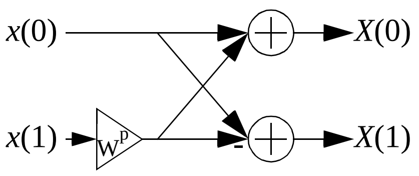
$$Temp_1 = c*(a + b)$$
$$Temp_2 = a*(d - c)$$
$$Temp_3 = b*(c + d)$$

Imag part = $Temp_1 + Temp_2$ $\{c(a + b) + a(d - c) = ad + cb\}$

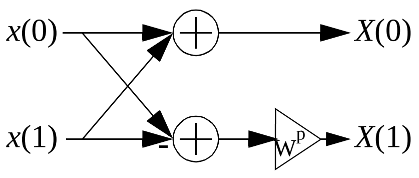
Real part = $Temp_1 - Temp_3$ $\{c(a + b) - b(c + d) = ac - bd\}$

11.23 We give only one example here: radix-2 butterfly element(including the twiddle factor multiplication).

The decimation-in-time and decimation-in-frequency butterfly is shown below:



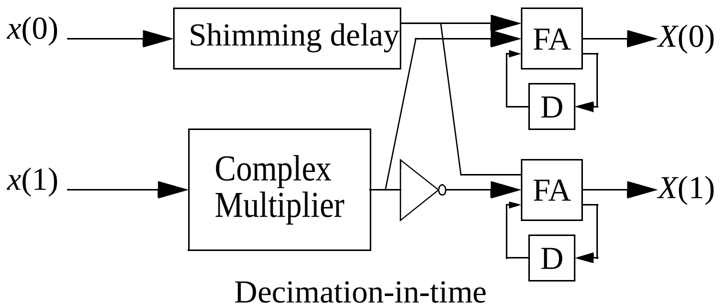
Decimation-in-time



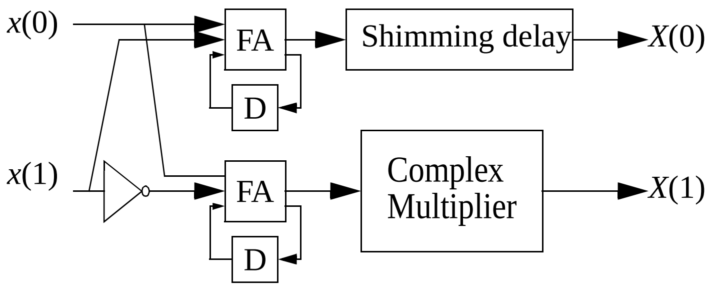
Decimation-in-frequency

Radix-2 Butterfly Elements

The twiddle factor multiplication is a complex multiplication. Since we know the coefficients in advance, we can use distributed arithmetic to reduce the number of real multiplications(See section 11.17). The complete shift-accumulator with s/p multiplier is shown in Figure 11.45, section 11.15. The radix-2 butterfly element can implemented as following.



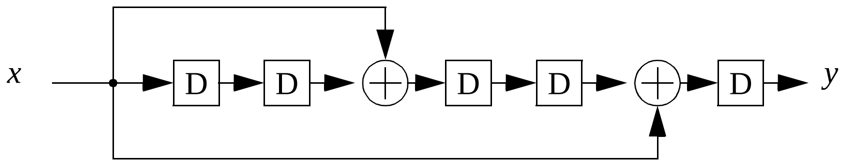
Decimation-in-time



Decimation-in-frequency

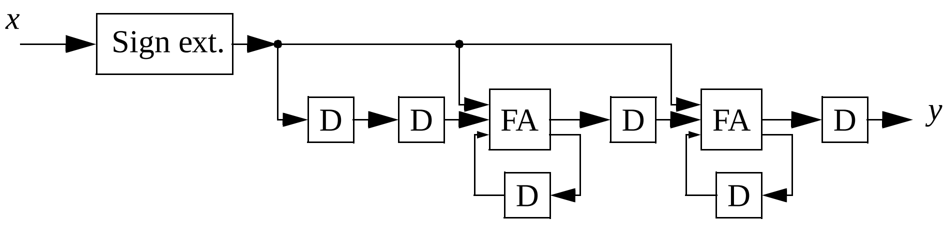
Butterfly Elements

11.25 (a) $a = 2^1(1 + 2^2(1 + 2^2))$, The multiplication can be realized as following.



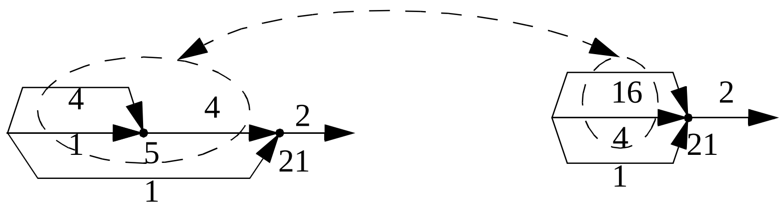
Multiplication with $a = 2^1(1 + 2^2(1 + 2^2))$

(b) The normal serial/parallel multipier for the same coeeficient can be realized as following, the simplification for the multiplier is omitted here.



Multiplication with $a = 2^1(1 + 2^2(1 + 2^2))$

(c) The relationship for these two multipliers can be expressed with graph representation.



(a) (b)

Graph representations

As we can see from the graph, they are equivalent.

11.30 Shift-and-add multiplier in VHDL

```
entity MULT is
        port(A_Port, B_Port: in bit_vector(3 downto 0);
             M_Out: out bit_vector(/ downto 0);
             CLK: in CLOCK;
             START: in BIT;
             DONE: out BIT);
end MULT;

architecture Shift_Mult of MULT is
begin
        process
             variable A, B, M: BIT_VECTOR;
             variable COUNT: INTEGER;
        begin
             wait until (START = 1);
             A := A_Port; COUNT := 0;
             B := B_Port; DONE <='  0'  ;
             M := B"0000";
             while (COUNT < 4) loop
                  if (A(0) = '1'   then
                       M := m + B;
                  end if;
                  A := SHR(A, M(0));
                  B := SHR(M, '0'    );
                  COUNT := COUNT + 1;
             end loop;
             M_Out <= M & A;
             DONE <= '1'    ;
        end process;
end SHIFT_MULT;
```

11.33 A bit-parallel implementation of complex multiplier using distributed arithmetic is implemented in Alcatel Mietec™ 0.35μm standard CMOS technology. In this bit-parallel complex multiplier, the shift-accumulator in Figure 11.48 is replaced with adder trees.
The main differences between the bit-parallel complex multiplier are: not need serial/parallel or parallel/serial interface is needed in bit-parallel implementation, the clock frequency is much more slower in bit-parallel implementation than that of bit-serial implementation, the glitch inside the bit-parallel multiplier is much larger than the bit-serial one, the accumulation of partial products can use, for example, tree structures for the bit-parallel implementation, the bit-serial implementation occupy obviously less area than the bit-parallel counterpart etc. The most important in the implementation style discussion is to know the advantages and disadvantages for bit-serial and bit-parallel (possible digit-serial) implementations.