Сложение, вычитание, умножение и сдвиг целых чисел

Обработка сигналов в цифровых устройствах включает в себя операции сложения/вычитания, умножения, деления, возведения в степень, взятия логарифмов и многие другие. Операции сложения/вычитания, умножения, а также, различные типы сдвигов, в современной программируемой логике легко реализуются в виде комбинационных схем. Выполнение более сложных операций требует перехода к последовательностным цифровым устройствам.

В этой лекции мы рассмотрим, как в программируемой логике выполняются простейшие арифметические операции над положительными и отрицательными целыми числами. Более подробно арифметика в фиксированной и плавающей точках будут рассмотрены в следующих лекциях.

Системы счисления

Перед тем, как перейти к вопросам реализации арифметических операций необходимо рассмотреть способы представления чисел. Здесь интересны системы счисления и представление чисел со знаком.

Система счисления представляет из себя набор правил и символов для записи числа. Человечество на протяжении истории своего развития пользовалось большим количеством разнообразных систем счисления. Так от древних римлян нам досталась римская система счисления (непозиционная), от шестидесятеричной системы счисления шумерского государства осталось деление часа на 60 минут, а минуты – на 60 секунд. В повседневной жизни и для научных расчетов сегодня обычно используется позиционная десятичная система счисления, а в вычислительной технике – позиционные двоичная, восьмеричная и шестнадцатеричная.

Позиционной (или поместной) называется такая система счисления, в которой у каждого символа в записи числа есть определенный вес, зависящий от позиции символа. Целое N-разрядное число в позиционной системе счисления с основанием b записывается следующим образом: $d_{N-1}\,d_{N-2}\,\ldots\,d_1\,d_0$. При этом само значение числа вычисляется так:

$$d_{N-1}\,d_{N-2}\,\ldots\,d_1\,d_0 = d_{N-1}\,b^{N-1} + d_{N-2}\,b^{N-2} + \ldots + d_1\,b + d_0.$$

Суммирование в приведенном выражении выполняется по модулю b, d_{N-1} – старший разряд числа (most significant bit, MSB), d_0 – младший разряд числа (least significant bit, LSB). Иногда может применяться запись числа от младшего разряда к старшему (слева направо).

Для записи нецелых чисел, применяется запятая для разделения целой и дробной частей числа. При этом основной принцип не меняется. (N+M)-разрядное число с N-разрядной целой частью и M-разрядной дробной записывается следующим образом:

$$d_{N-1}\,d_{N-2}\,\ldots\,d_1\,d_0\,,\,d_{-1}\,d_{-2}\,\ldots\,d_{-M} = d_{N-1}\,b^{N-1} + d_{N-2}\,b^{N-2} + \ldots + d_1\,b + d_0 + d_{-1}\,b^{-1} + d_{-2}\,b^{-2} + \ldots + d_{-M}\,b^{-M}$$

В **десятичной (dec)** системе счисления[1] для записи чисел используются цифры 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, основание b = 10.

В **двоичной (bin)** системе счисления для записи чисел используются цифры 0 и 1, основание b = 2.

В **восьмеричной (oct)** системе счисления для записи чисел используются цифры 0, 1, 2, 3, 4, 5, 6, 7, основание b = 8.

[1] Далее для краткости не будем указывать, что система счисления является позиционной.

В **шестнадцатеричной (hex)** системе счисления для записи чисел используются цифры 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f, основание $b = 16$.

Одно и то же число может быть записано в разных системах счисления, т.к. система счисления определяет только способ записи числа.

Преобразования чисел из десятичной системы счисления в двоичную, восьмеричную или шестнадцатеричную выполняются путем последовательного деления исходного числа на 2, 8 или 16 соответственно и записывании остатков от деления на каждом шаге. Значение символа на *k*-м шаге равно остатку от деления целой части числа, полученной на предыдущем шаге. Процедура продолжается до тех пор, пока целая часть десятичного числа не обратится в ноль. При этом вычисляются значения символов начиная с младшего. Например, переведем число 48815_{dec} в шестнадцатеричную систему счисления (таблица 1) и число 85_{dec} – в двоичную систему счисления (таблица 2). $48815_{dec} = beaf_{hex}$, $85_{dec} = 1010101_{bin}$.

Таблица 1. Перевод числа 48815 из десятичной системы счисления в шестнадцатеричную

Номер шага	Что делим	Целая часть от деления на 16	Остаток от деления на 16	Полученный символ
1	48815	3050	15	f
2	3050	190	10	a
3	190	11	14	e
4	11	0	11	b

Таблица 2. Перевод числа 85 из десятичной системы счисления в двоичную

Номер шага	Что делим	Целая часть от деления на 2	Остаток от деления на 2	Полученный символ
1	85	42	1	1
2	42	21	0	0
3	21	10	1	1
4	10	5	0	0
5	5	2	1	1
6	2	1	0	0
7	1	0	1	1

Преобразования чисел из двоичной в восьмеричную или шестнадцатеричную системы счисления и обратно очень простые.

Перевод двоичного числа в восьмеричную (шестнадцатеричную) систему счисления:

- в исходном числе сгруппировать символы в группы из трех (четырех) символов;
- каждой группе из трех (четырех) символов поставить в соответствие восьмеричный (шестнадцатеричный) символ.

Например, имеем двоичное число 1101111010101101_{bin}. Переведем его в шестнадцатеричную систему счисления. Для этого сгруппируем символы в заданном числе в группы по четыре символа: 1101 1110 1010 1101. Далее каждой группе поставим в соответствие шестнадцатеричный символ: $1101_{bin} = 13_{dec} = d_{hex}$, $1110_{bin} = 14_{dec} = e_{hex}$ и т. д. Имеем $1101111010101101_{bin} = dead_{hex}$.

Перевод восьмеричного (шестнадцатеричного) числа в двоичную систему счисления:

- в исходном числе каждый восьмеричный (шестнадцатеричный) символ должен быть заменен группами из соответствующих трех (четырех) двоичных символов.

Например, имеем два восьмеричных числа 171_{oct} и 133_{oct}. Эти числа описывают образующие полиномы одного из наиболее часто применяющихся в системах связи и передачи данных помехоустойчивый код. Переведем их в двоичную систему счисления. Для этого каждому восьмеричному символу поставим в соответствие необходимую комбинацию из трех двоичных символов: $171_{oct} = 001\ 111\ 001_{bin}$, $133_{oct} = 001\ 011\ 011_{bin}$.

Представление двоичных чисел со знаком. Прямой, обратный и дополнительный коды

Для представления чисел со знаком[2] используются прямой, обратный и дополнительный коды. В английской литературе их называют соответственно sign-magnitude, 1's complement и 2's complement.

Общее представление чисел со знаком с *N*-разрядной целой частью и *M*-разрядной дробной представлено на рис. 1. Во всех кодах для указания знака числа выделяется один разряд. Далее будем считать, что в качестве знакового выбран старший (крайний левый) разряд. Если знаковый разряд равен 0, то число положительное, если 1 – отрицательное. Остальные разряды используются для записи модуля числа. Во всех кодах неотрицательные числа записываются одинаково. Коды различаются только представлением отрицательных чисел, т.е. записью модуля числа. Отметим, что в любом способе представления чисел подразумевается заданная разрядность. Например, 32-х битные числа.

Рис. 1. Общее представление числа со знаком

В **прямом коде** кодирование двоичного числа выполняется по формуле:

$$X_{binПК} = \begin{cases} X, \ X \geq 0 \\ 2^{N-1} + |X|, \ X < 0 \end{cases}.$$

Перевод двоичного числа, представленного в прямом коде, в десятичную систему счисления выполняется следующим образом:

$$X_{dec} = (1 - 2d_{N-1}) \sum_{k=-M}^{N-2} d_k 2^k ,$$

где d_k – цифры в записи числа X.

В таблице 3 приведены примеры нескольких двоичных чисел, записанных в прямом коде. Красным цветом выделен знаковый разряд. Из таблицы 3 видно, что противоположные числа различаются *только* в знаковом разряде.

Таблица 3. Пример представления двоичных чисел в прямом 8-разрядном коде

Десятичное число	Двоичное число в прямом коде	Двоичное число
3	00000011	11
–3	10000011	–11
10	00001010	1010
–10	10001010	–1010
100	01100100	1100100
–100	11100100	– 1100100
0	00000000	0
–0	10000000	–0

[2] В любой позиционной системе счисления.

Представление чисел в прямом коде соответствует привычному представлению чисел в научных расчетах, когда, фактически сначала записывается знак числа, а затем – его модуль. Отметим, что представление чисел в прямом коде возможно для любых систем счисления.

В машинных вычислениях запись чисел со знаком в прямом коде неудобна, т.к. при выполнении арифметических операций требуется особая обработка знакового разряда (он не имеет веса). Также в прямом коде есть два нуля: “положительный” и “отрицательный”. Для обработки “отрицательного” нуля также требуются специальные решения в архитектуре арифметического процессора.

В отличие от прямого в **обратном коде** при записи отрицательного числа его модуль записывается в инвертированном виде. Т.е. для изменения знака числа необходимо проинвертировать все его разряды, включая знаковый. В таблице 4 приведены примеры нескольких чисел, записанных в обратном коде. Красным цветом выделен знаковый разряд.

Таблица 4. Пример представления чисел в обратном 8-разрядном коде

Десятичное число	Двоичное число в обратном коде
3	00000011
–3	11111100
10	00001010
–10	11110101
100	01100100
–100	10011011
0	00000000
–0	11111111

Из таблицы 4 видно, что противоположные числа получаются друг из друга простой инверсией всех разрядов. Интересно, что также, как и у прямого, у обратного кода есть “положительный” и “отрицательный” нули, а сумма двух противоположных чисел дает –0. Существование таких нулей приводит к дополнительному усложнению архитектуры арифметического процессора.

Наибольшую популярность для представления чисел со знаком в машинных вычислениях имеет **дополнительный код**. Для записи отрицательного числа в дополнительном коде его модуль инвертируется и к результату добавляется 1. Пусть имеется неотрицательное двоичное число A. Число B, противоположное A, – это такое число, которое в сумме с A будет давать 0. Для нахождения B необходимо инвертировать все разряды числа A и к получившемуся числу добавить 1. Старший разряд при этом знаковый: 0 – число положительное, 1 – отрицательное. Например, пусть задано число $17_{dec} = 10001_{bin}$, разрядность чисел – 8 бит. Необходимо записать число –17. Число 17_{dec} в двоичном виде в 8-и разрядном формате записывается так: 00010001. Далее обратим все разряды: 11101110. К получившемуся числу добавим 1: 11101111. Таким образом, число –17 в двоичном виде в дополнительном коде имеет вид 11101111. Несложно убедиться, что 11101111 + 00010001 = 100000000. Но старшая единица выходит за пределы заданной разрядности и поэтому не учитывается в дальнейших расчетах.

В таблице 5 представлены примеры представления чисел в обратном и дополнительном 8-разрядных кодах.

Таблица 5. Пример представления чисел в обратном и дополнительном 8-разрядных кодах

Десятичное число	Двоичное число в обратном коде	Двоичное число в дополнительном коде
3	00000011	00000011
–3	11111100	11111101
10	00001010	00001010
–10	11110101	11110110
100	01100100	01100100

–100	10011011	10011100
0	00000000	00000000
–0	11111111	00000000

Применение дополнительного кода существенно упрощает архитектуру арифметического процессора. Это связано с тем, что сложение и вычитание в дополнительном коде выполняется одинаково, т.е. одними и теми же цепями (см. ниже). Также в дополнительном коде существует только один 0 (положительный).

Важным недостатком дополнительного кода является разное количество возможных ненулевых положительных и отрицательных чисел. В дополнительном коде не существует числа, противоположного наименьшему числу. Например, при использовании 8 разрядов, наименьшее представимое число $-128_{dec} = 10000000_{bin}$, а наибольшее – только $127_{dec} = 01111111_{bin}$. Это значит, что изменение знака числа требует дополнительной проверки, не инвертируется ли число –128.

Важно понимать, что как только выбрана разрядность для задания двоичного числа (а в цифровых устройствах она всегда ограничена), автоматически выбирается и диапазон возможных значений чисел. В случае беззнаковых целых будем иметь диапазон от 0 до $(2^N - 1)$, в случае знаковых целых, представленных в дополнительном коде, – от $-2^{(N-1)}$ до $2^{(N-1)} - 1$. Во втором случае один разряд (старший) расходуется на указание знака числа.

Далее в курсе для записи двоичных чисел при выполнении арифметических операций будет использоваться только дополнительный код.

Изменение разрядности чисел в дополнительном коде

При выполнении вычислений в цифровых устройствах часто приходится оперировать числами с разными разрядностями. В этом случае может возникнуть неопределенность в интерпретации разрядов числа с меньшей разрядностью: будут ли они интерпретированы средствами проектирования как старшие или как младшие. Для того чтобы избежать возможных неопределенностей, предотвратить возможное переполнение (см. следующий пункт), а также для улучшения читаемости кода и прогнозируемости результатов его работы иногда полезно явно увеличивать разрядности операндов.

Предположим, имеется некоторое число X с разрядностью N. Необходимо увеличить его разрядность до $K > N$. В зависимости от того, что заранее известно о знаке X, возможны следующие ситуации.

1. $X \geq 0$. В этом случае необходимо добавить $(K - N)$ нулей слева от числа. Например, пусть $X = 0110$ (6_{dec}) и разрядность необходимо увеличить до 8. Необходимо добавить четыре нуля слева от X: $X = \mathbf{0000}0110$.

2. $X < 0$. В этом случае необходимо добавить $(K - N)$ единиц слева от числа. Например, пусть $X = 1110$ (-2_{dec}) и разрядность необходимо увеличить до 8. Необходимо добавить четыре единицы слева от X: $X = \mathbf{1111}1110$.

3. Заранее ничего не известно о знаке X: число может быть либо положительным, либо отрицательным. В этом случае нельзя использовать предыдущие подходы, т.к. можно исказить число. Слева от числа необходимо добавить $(K - N)$ единиц, если число отрицательное, или столько же нулей, если число неотрицательное. Для выполнения такой операции необходимо слева от числа $(K - N)$ раз скопировать знаковый разряд.

При выполнении описанных операций значение числа не меняется, меняется только его разрядность.

Сложение

Сложение *N*-разрядных двоичных чисел выполняется так же, как и десятичных: числа записываются друг под другом, далее выполняется поразрядное сложение с учетом переноса с предыдущего разряда, запись результирующей цифры в необходимый разряд и формирование цифр переноса. При сложении двоичных чисел бит переноса формируется, когда результатом сложения для текущего разряда являются 2 или 3 (в десятичном представлении).

Таблица 6. Пример сложения двух 4-х разрядных двоичных чисел

C	0	1	1	0
X	1	0	0	1
Y	0	0	1	1
Z	**1**	**1**	**0**	**0**

В таблице 6 приведен пример сложения двух 4-х разрядных двоичных чисел. В примере *C* – бит переноса, значение которого формируется при суммировании каждой пары разрядов *X* и *Y*. Для пары младших разрядов бит переноса равен нулю.

Таким образом, при сложении двух *N*-разрядных двоичных чисел *N* раз выполняются следующие однотипные операции: сложение трех битов $z_i = x_i \oplus y_i \oplus c_i$ и формирование бита переноса $c_{i+1} = x_i \times y_i + x_i \times c_i + y_i \times c_i$. Здесь знаком + обозначена логическая сумма. Комбинационное цифровое устройство, выполняющее эти две операции, называется полным сумматором. Полный сумматор (full adder, FA) имеет три входа и два выхода (см. рис. 2).

В таблице 7 представлена таблица истинности полного одноразрядного сумматора.

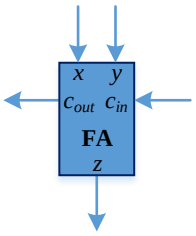

Рис. 2. Одноразрядный полный сумматор

Таблица 7. Таблица истинности полного сумматора

x	***y***	c_{in}	c_{out}	***z***
0	0	0	**0**	**0**
0	0	1	**0**	**1**
0	1	0	**0**	**1**
0	1	1	**1**	**0**
1	0	0	**0**	**1**
1	0	1	**1**	**0**
1	1	0	**1**	**0**
1	1	1	**1**	**1**

Для сложения *N*-разрядных чисел требуется *N* одноразрядных полных сумматоров, соединенных последовательно по входам и выходам бита переноса (см. рис. 3).

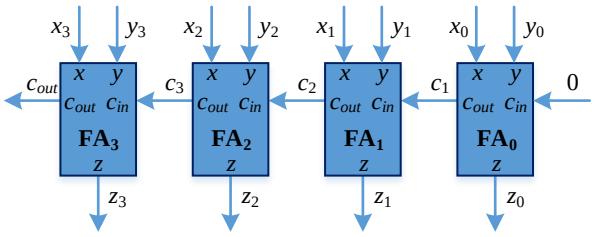

Рисунок 3. Параллельный четырехразрядный сумматор, собранный из полных одноразрядных сумматоров. Бит cout является флагом переполнения сумматора

Несмотря на то, что все полные сумматоры в параллельном сумматоре на рис. 3 работают одновременно, быстродействие такого сумматора ограничено временем распространения битов переноса по цепям переноса справа налево. Другими словами, при подаче на вход параллельного сумматора двух новых слагаемых, сумма на его выходе появится только через определенный интервал времени, определяемый временем прохождения битов переноса через весь сумматор. Чем больше разрядность операндов, тем больше времени необходимо для их сложения.

В случае, когда для реализации сумматора доступны таблицы истинности с количеством входов больше двух (например, четыре, шесть и т.д.), можно реализовывать полный сумматор, который складывает сразу несколько разрядов двоичных чисел (два, три и т.д.) и тем самым уменьшать количество цепей переноса и увеличивать быстродействие сумматора. В качестве примера в таблице 8 представлена таблица истинности полного сумматора двух разрядных чисел (см. рис. 4).

Таблица 8. Таблица истинности полного сумматора двухразрядных чисел

c_{in}	x_1	x_0	y_1	y_0	c_{out}	z_1	z_0	***Результат, dec***
0	0	0	0	0	**0**	**0**	**0**	**0**
0	0	0	0	1	**0**	**0**	**1**	**1**
0	0	0	1	0	**0**	**1**	**0**	**2**
0	0	0	1	1	**0**	**1**	**1**	**3**
0	0	1	0	0	**0**	**0**	**1**	**1**
0	0	1	0	1	**0**	**1**	**0**	**2**
0	0	1	1	0	**0**	**1**	**1**	**3**
0	0	1	1	1	**1**	**0**	**0**	**4**
0	1	0	0	0	**0**	**1**	**0**	**2**
0	1	0	0	1	**0**	**1**	**1**	**3**
0	1	0	1	0	**1**	**0**	**0**	**4**
0	1	0	1	1	**1**	**0**	**1**	**5**
0	1	1	0	0	**0**	**1**	**1**	**3**
0	1	1	0	1	**1**	**0**	**0**	**4**
0	1	1	1	0	**1**	**0**	**1**	**5**
0	1	1	1	1	**1**	**1**	**0**	**6**
1	0	0	0	0	**0**	**0**	**1**	**1**
1	0	0	0	1	**0**	**1**	**0**	**2**
1	0	0	1	0	**0**	**1**	**1**	**3**
1	0	0	1	1	**1**	**0**	**0**	**4**
1	0	1	0	0	**0**	**1**	**0**	**2**
1	0	1	0	1	**0**	**1**	**1**	**3**
1	0	1	1	0	**1**	**0**	**0**	**4**
1	0	1	1	1	**1**	**0**	**1**	**5**
1	1	0	0	0	**0**	**1**	**1**	**3**
1	1	0	0	1	**1**	**0**	**0**	**4**
1	1	0	1	0	**1**	**0**	**1**	**5**
1	1	0	1	1	**1**	**1**	**0**	**6**
1	1	1	0	0	**1**	**0**	**0**	**4**
1	1	1	0	1	**1**	**0**	**1**	**5**
1	1	1	1	0	**1**	**1**	**0**	**6**
1	1	1	1	1	**1**	**1**	**1**	**7**

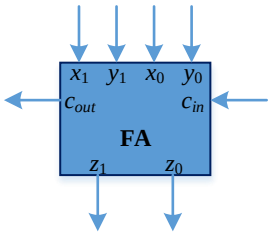

Рисунок 4. Полный сумматор двухразрядных чисел

Для повышения быстродействия параллельного сумматора известны различные техники, однако в современных FPGA для цепей переноса выделяются специальные линии, скорость распространения сигналов по которым чрезвычайно высока. Это позволяет реализовывать многоразрядные сумматоры с высоким быстродействием, не усложняя архитектуру сумматора.

Вычитание

Для вычитания чисел, записанных в дополнительном коде, применяется следующая простая последовательность действий:

1) Найти число, дополнительное к вычитаемому (т.е. инвертировать все его разряды и прибавить 1).
2) Сложить результат из 1) с уменьшаемым. Для этого можно использовать, например, сумматоры из предыдущего пункта.

Для того, чтобы упростить операцию вычитания, добавление 1 можно перенести из первого шага во второй. Для этого во втором шаге на первый (самый правый) полный сумматор в параллельном сумматоре на вход бита переноса необходимо подать 1, а не 0 как при сложении чисел.

Сдвиг

При выполнении операций сдвига биты в двоичном числе смещаются влево или вправо на одну (shifter) или несколько (barrel shifter) позиций.

Различают следующие виды операций сдвига:

- Логический сдвиг.
- Арифметический сдвиг
- Циклический сдвиг

При циклическом сдвиге вправо (влево) биты, вышедшие за пределы разрядности числа добавляются к исходному числу слева (справа).

При логическом и арифметическом сдвигах вправо (влево) биты, вышедшие за разрядность числа удаляются. Различие заключается в том, что при арифметическом сдвиге считается, что сдвигаемое число знаковое, а при логическом – без знаковое. При логическом сдвиге вправо к числу слева добавляются нули, а при арифметическом – повторяется старший (знаковый) разряд исходного сдвигаемого числа. Арифметический и логический сдвиги влево одинаковы: при сдвиге к числу справа добавляются нули.

Таблица 9. Выполнение различных сдвигов на один разряд

Исходное число	11101100
Логический сдвиг влево	11011000
Арифметический сдвиг влево	11011000
Циклический сдвиг влево	11011001
Исходное число	11101100
Логический сдвиг вправо	01110110
Арифметический сдвиг вправо	11110110
Циклический сдвиг вправо	01110110

Арифметический сдвиг числа влево на один двоичный разряд эквивалентен умножению числа на 2, а сдвиг на один двоичный разряд вправо соответствует делению числа на 2.

Умножение

Умножение двоичных *N*-разрядных чисел выполняется так же, как и десятичных – “в столбик”. На каждом шаге операции умножения выполняется расчет промежуточных результатов x_i AND y_j для всех возможных значений i и j, затем промежуточные результаты определенным образом складываются. Ниже представлена процедура умножения двух четырехразрядных двоичных чисел.

*Таблица 10. Пример умножения 4-*х разрядных чисел “в столбик”

X			x_3	x_2	x_1	x_0

Y					×	y_3	y_2	y_1	y_0
						$x_3 y_0$	$x_2 y_0$	$x_1 y_0$	$x_0 y_0$
				+	$x_3 y_1$	$x_2 y_1$	$x_1 y_1$	$x_0 y_1$	
			+	$x_3 y_2$	$x_2 y_2$	$x_1 y_2$	$x_0 y_2$		
		+	$x_3 y_3$	$x_2 y_3$	$x_1 y_3$	$x_0 y_3$			
P	p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0	

В современных ПЛИС возможно два способа реализации умножителей: на логике общего назначения или на встроенных умножителях.

В первом способе умножители представляют из себя несколько сумматоров (в рассмотренном примере их три), на вход которых подаются промежуточные результаты умножения (операции AND) соответствующих разрядов перемножаемых чисел.

Во втором способе умножитель представляет из себя заранее (на этапе создания ПЛИС) оптимизированный и выращенный в кристалле элемент, т.е. простой ASIC в ПЛИС. Такие элементы позволяют выполнять не только операцию умножения, но и сложения, сдвига. Обычно встроенные умножители размещаются в ПЛИС по столбцам и позволяют очень быстро выполнять операции умножения и сложения с накоплением, необходимые при реализации фильтров.

Изменение разрядности при выполнении сложения и умножения

При выполнении арифметических операций при конечной и *одинаковой* разрядности операндов и результата всегда есть ненулевая вероятность переполнения. Например, при сложении или умножении двух знаковых чисел, максимальных по модулю, получится число, для записи которого необходимо больше разрядов, чем у операндов.

Переполнение – это негативный эффект, с которым борются либо увеличением разрядности для представления результата, либо применяя различные способы округления. Об этом будет рассказано в одной из последующих лекций.

Пример 1. Умножение на рациональное число и деление на целое

При выполнении математических операций при обработке сигналов часто возникает необходимость поделить входное число на некоторую целую константу или, в общем случае, умножить входное число на дробь вида m/n, где m и n – целые положительные числа. Т.е. необходимо выполнить следующие операции: $y = x/n$ или $y = x \cdot m/n$, где x – входное N-разрядное число с фиксированной точкой. Например, умножение на 3/5, 5/7, деление на 3, 5, 7, и т.п. Так или иначе, обе операции подразумевают выполнение деления на целую константу. Но операция деления – это достаточно дорогая операция для FPGA, для нее нет аппаратной поддержки и она может требовать много ресурсов (десятки и сотни таблиц истинности) и выполняться за несколько тактов для получения результата.

Тем не менее, при выполнении рассматриваемых операций можно обойтись только умножением на заранее рассчитанную константу. Ключевым моментом здесь является ограниченность разрядностей входного и выходного сигналов, из-за чего результат выполнения операции будет представлен в общем случае с некоторой ошибкой, обусловленной ограниченной точностью представления чисел с фиксированной точкой. Это значит, что числа $1/n$ и m/n можно представить в виде некоторого числа с фиксированной точкой с таким числом разрядов, что результат их умножения на любое возможное число x будет давать правильный результат в выбранной разрядности выходного сигнала.

Рассмотрим для примера задачу умножения 5-разрядного беззнакового целого[3] числа x на константу 2/3. Выделим для записи результата 7 бит, два младших разряда – дробные.

[3] Приведенные рассуждение справедливы и для общего случая обработки знаковых чисел с фиксированной точкой.

Необходимо найти такое число C, которое, представленное с фиксированной точкой, для любого возможного 5-разрядного числа x даст правильный результат с учетом выбранной разрядности результата. Другими словами, необходимо подобрать такую аппроксимацию C числа 2/3 с фиксированной точкой, что результат для заданных разрядностей будет корректным.

Далее необходимо выполнить следующие шаги.

1. Для каждого возможного входного числа x от 0 до 31 посчитать результат операции $x \cdot 2/3$ (см. второй столбец табл. 1) с достаточно большой точностью (в плавающей точке, например, в Excel или Matlab).

2. Зная число разрядов для записи результата и положение точки в нем, можно посчитать приближенное значение операции $x \cdot 2/3$ для заданного числа разрядов и положения точки. Для этого проще всего умножить x на 2^2 (на дробную часть выделено 2 разряда), затем результат умножить на 2/3 и отбросить дробную часть (см. третий столбец табл. 1, операция floor – округление в сторону меньшего числа). Если перевести полученные целые числа в двоичный вид и разместить запятую слева от двух младших разрядов, то получим требуемый результат умножения x на 2/3 (см. четвертый столбец табл. 1).

Таким образом, в пятом столбце табл. 1 записаны все возможные результаты умножения пятиразрядного целого неотрицательного числа x на константу 2/3, при чем результат умножения записывается в 7 разрядное число с фиксированной точкой, в котором два младших разряда – дробные.

3. Далее необходимо с достаточной точностью записать в двоичном виде число 2/3. Под достаточной точностью подразумевается такое представление константы, что при умножении ее на любое число из первого столбца таблицы получится соответствующее число из четвертого (в двоичной системе счисления) и пятого (в десятичной системе счисления) столбца табл. 1.

Таблица 1. Результат операции $x \cdot 2/3$ с плавающей точкой

x	$x \cdot 2/3$	floor($x \cdot 4 \cdot 2/3$)	floor($x \cdot 4 \cdot 2/3$)/4_{bin}	floor($x \cdot 4 \cdot 2/3$)/4_{dec}	Модуль ошибки
0	0,0000	0	00000,00	0,00	0,0000
1	0,6667	2	00000,10	0,50	0,1667
2	1,3333	5	00001,01	1,25	0,0833
3	2,0000	8	00010,00	2,00	0,0000
4	2,6667	10	00010,10	2,50	0,1667
5	3,3333	13	00011,01	3,25	0,0833
6	4,0000	16	00100,00	4,00	0,0000
7	4,6667	18	00100,10	4,50	0,1667
8	5,3333	21	00101,01	5,25	0,0833
9	6,0000	24	00110,00	6,00	0,0000
10	6,6667	26	00110,10	6,50	0,1667
11	7,3333	29	00111,01	7,25	0,0833
12	8,0000	32	01000,00	8,00	0,0000
13	8,6667	34	01000,10	8,50	0,1667
14	9,3333	37	01001,01	9,25	0,0833
15	10,0000	40	01010,00	10,00	0,0000
16	10,6667	42	01010,10	10,50	0,1667
17	11,3333	45	01011,01	11,25	0,0833
18	12,0000	48	01100,00	12,00	0,0000
19	12,6667	50	01100,10	12,50	0,1667
20	13,3333	53	01101,01	13,25	0,0833
21	14,0000	56	01110,00	14,00	0,0000
22	14,6667	58	01110,10	14,50	0,1667
23	15,3333	61	01111,01	15,25	0,0833
24	16,0000	64	10000,00	16,00	0,0000

25	16,6667	66	10000,10	16,50	0,1667
26	17,3333	69	10001,01	17,25	0,0833
27	18,0000	72	10010,00	18,00	0,0000
28	18,6667	74	10010,10	18,50	0,1667
29	19,3333	77	10011,01	19,25	0,0833
30	20,0000	80	10100,00	20,00	0,0000
31	20,6667	82	10100,10	20,50	0,1667

Число 2/3 = 0,(6). В двоичном представлении с фиксированной точкой его можно приближенно записать в числах с разной разрядностью с разной абсолютной ошибкой как представлено в табл. 2. Из табл. 2 видно, что абсолютная ошибка уменьшается с увеличением числа разрядов, отведенных для записи числа 2/3.

Таблица 2. Приближенное представление числа 2/3 в двоичном виде с фиксированной точкой

Число разрядов	Приближенное значение в двоичной системе счисления, C_{bin}	Приближенное значение в десятичной системе счисления, C_{dec}	$\approx\lvert C_{dec} - 2/3\rvert$
2	0,1	0,5	0,1667
3	0,11	0,75	0,0833
4	0,101	0,625	0,0417
5	0,1011	0,6875	0,0208
6	0,10101	0,65625	0,0104
7	0,101011	0,671875	0,0052
8	0,1010101	0,6640625	0,0026
9	0,10101011	0,66796875	0,0013

Для решения поставленной задачи подходит приближенное значение числа 2/3 с 8 знаками после запятой $10101011_{bin} = 171_{dec}$, причем 171/256 ≈ 2/3. Это значит, что для получения результата необходимо умножить входное число x на целое число 171, разделить на $256 = 2^8$ (т.е. просто переместить точку на 8 разрядов влево) и от получившегося числа взять старшие семь разрядов.

Например, умножим число 27 на 2/3.

- $27 \cdot 171 = 4617_{dec} = 1\ 0010\ 0000\ 1001_{bin}$.
- $4617/256 = 18{,}03515625_{dec} = 1\ 0010,\ 0000\ 1001_{bin}$
- Берем старшие 7 разрядов результата: 1 0010, 00. Результат совпадает с требуемым результатом из столбца 4 табл. 1.

Следует отметить, что для деления двух чисел в общем случае необходимо использовать специальные алгоритмы, типа деления в столбик или итеративного. Обычно операция деления присутствует в виде IP-ядер в библиотеках среды разработки FPGA.

Пример 2. Модуль комплексного числа

Обработка сигналов нередко предполагает выполнение различных операций над комплексными числами. Так комплексные числа появляются при обработке сигналов связи в квадратурном представлении, обработке изображений и др.

Достаточно “неудобной” при этом является операция нахождения модуля комплексного числа $z = x + jy$: $|z| = \sqrt{x^2 + y^2}$, которая требует нахождения корня квадратного от некоторой величины, а также двух умножений вещественных чисел. Для точного (в заданной разрядности) нахождения модуля комплексного числа можно применять табличную реализации функции или ее аппроксимацию рядом Тейлора. Однако на практике удобно использовать следующую аппроксимацию:

$$
\begin{aligned}
a &= \max\{abs(x), abs(y)\}, \\
b &= \min\{abs(x), abs(y)\}, \\
|z| &= 0{,}875a + 0{,}5b.
\end{aligned} \tag{1}
$$

Т.е сначала находятся абсолютные значения вещественной и мнимой частей заданного комплексного числа, затем большее из них умножается на константу 0,875, а меньшее – на 0,5. В итоге результаты умножений на константы складываются.

Например, пусть $z = 115 - j77$. Значение $|z|$, рассчитанное по определению $|z| \approx 138{,}398$. Значение $|z|$, рассчитанное по приближенной формуле $|z| = 0{,}875 \cdot 115 + 0{,}5 \cdot 77 = 139{,}125$. Относительная ошибка составила $|139{,}125 - 138{,}398|/138{,}398 \approx 0{,}005 < 1\%$.

Особенности FPGA

В современных FPGA присутствуют встроенные DSP элементы (как бы миниатюрные ASIC), которые представляют из себя аппаратную реализацию операции MAC. Другими словами, встроенные DSP элементы умеют эффективно (быстро, без затрат логики общего назначения, с низким энергопотреблением) выполнять операции сложения, умножения и умножения с накоплением.

DSP элемент представляет из себя полноценное арифметико-логическое устройство и состоит из трех последовательно соединенных блоков: сумматор/вычитатель (так называемый, пресумматор, preadder), умножитель, сумматор/вычитатель/аккумулятор. Такая структура позволяет эффективно реализовывать типовую операцию умножения с накоплением (multiply-accumulate, MAC), на основе которой реализуются фильтры с конечной импульсной характеристикой.

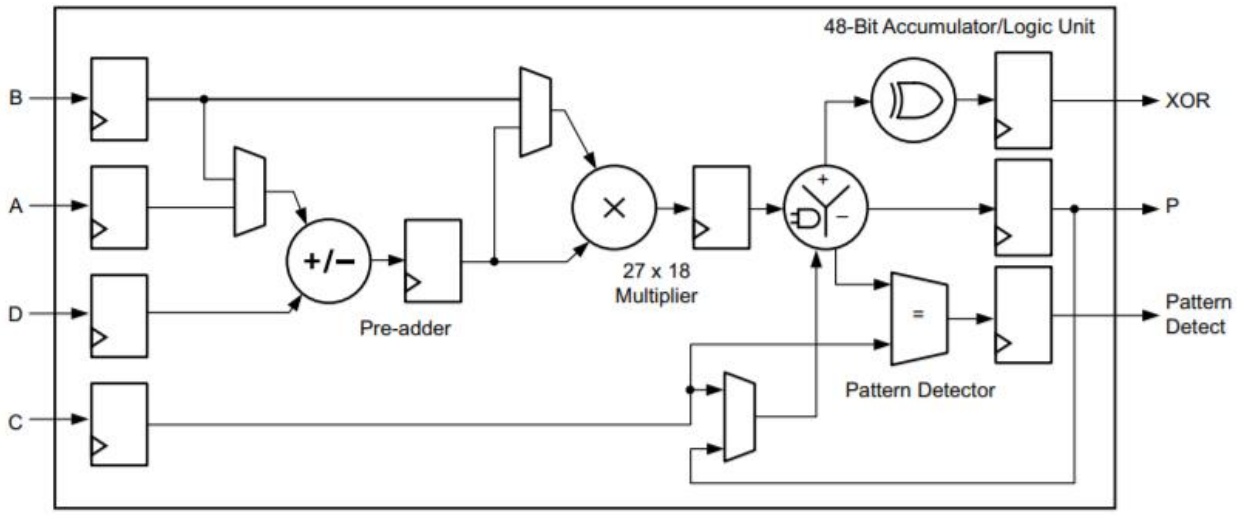

Рисунок 5. Структура DSP элемента Xilinx FPGA серии Ultrascale[4]

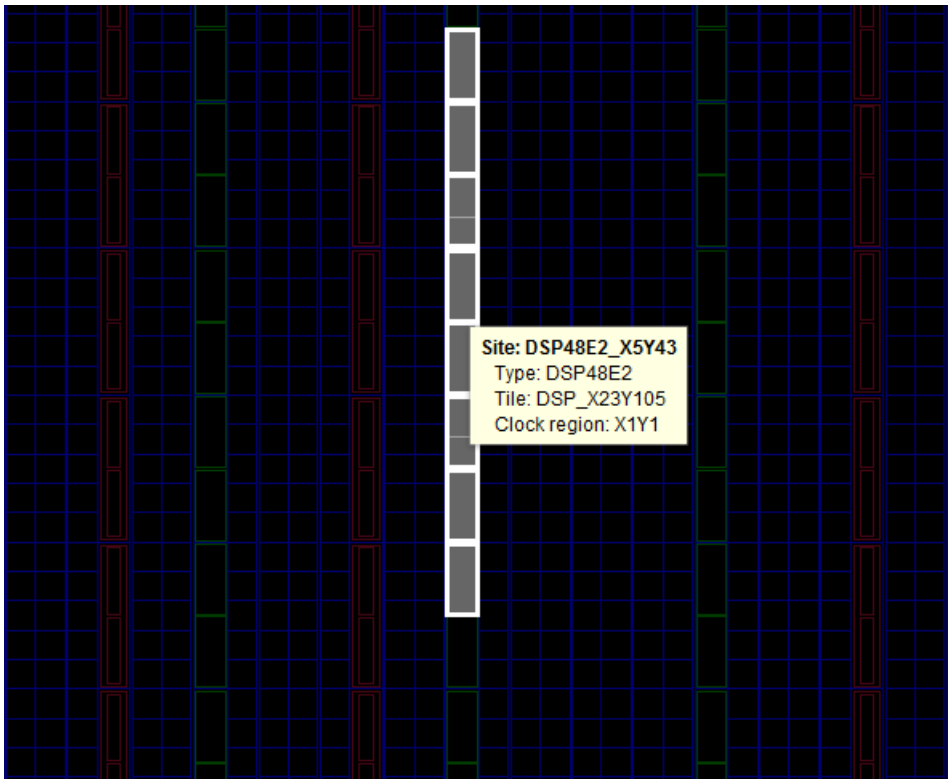

Рисунок 6. Размещение DSP элементов внутри FPGA по столбцам (зеленые блоки – DSP элементы)

Также DSP элементы позволяют реализовывать операции поиска последовательностей (pattern detector) и сдвигающего регистра с управляемой величиной сдвига (barrel shifter).

DSP элементы располагаются в FPGA в столбцах, что позволяет создавать фильтры, поддерживающие очень высокие системные частоты (~500 МГц). В топовых FPGA Xilinx может быть несколько тысяч DSP элементов (до 2800).

[4] UltraScale Architecture DSP Slice. User Guide UG579