УДК 51

РАБОТА СУММАТОРА НА ОСНОВЕ БУЛЕВЫХ ФУНКЦИЙ

Николаева Е.А., к.м.-ф.н., доцент
Коковин В.А., студент гр. ПИб-221, II курс
Кузбасский государственный технический университет
имени Т.Ф. Горбачева
г. Кемерово

Существует множество систем счисления. Современный человек использует в повседневной жизни десятичную, во времена шумерской цивилизации использовали шестидесятеричную. Компьютер и другая электроника использует же двоичную систему счисления. Данный факт дает нам возможность представить работу электронных устройств и ЭВМ с помощью булевых функций, где 1 – наличие электронов, а 0 – их отсутствие.

Цель работы: изучить информацию о работе электронных устройств, наглядно продемонстрировать её на основе схем и алгоритмов.

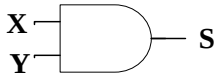
Актуальность работы: данная исследовательская работа будет полезна людям, которые хотят познакомиться с логикой поведения простых электронных устройств, таких как сумматор.

В литературе есть много информации на эту тему, но сложность их изложения и недостающая в них сведения являются фактором на проведение этого исследования.

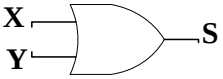
Примечания:

1. Для изображения схем используется стандарт ANSI.
2. Для написания алгоритма используется язык программирования Python.
3. Примем в Python: “True” = “1”, “False” = “0”, оператор “and” = “конъюнкция”, оператор “or” = “дизъюнкция”, оператор “not” = “отрицание”.
4. Математические действия производятся над двоичными числами, чтоб не углубляться в тему перевода чисел из одних систем счисления в другие.

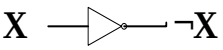
Конъюнкция, операция И. Схема данной операции выглядит так:



Дизъюнкция, операция ИЛИ. Схема данной операции выглядит так:



Отрицание, операция НЕ. Схема данной операции выглядит так:

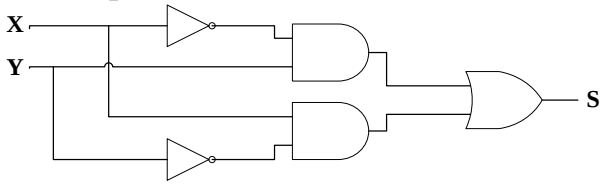


Реализация работы четверти сумматора (одноразрядного).

Четверть сумматор складывает числа, но без перехода числа в следующий разряд.

$$S = f(X, Y) = (\neg X \wedge Y) \vee (X \wedge \neg Y)$$

Схема четверть сумматора:



Программная запись функции f, назовем её в программе для дальнейшего использования quarter_sum, которая возвращает значение S:

```
def quarter_sum(X, Y):
  return (not X and Y) or (X and not Y)
```

Подаются значения X и Y и на выходе возвращается S – сумма X и Y.

Таблица логических операций:

X	Y	¬X	¬Y	¬X∧Y	X∧¬Y	(¬X∧Y)∨(X∧¬Y)=S
0	0	1	1	0	0	0
0	1	1	0	1	0	1
1	0	0	1	0	1	1
1	1	0	0	0	0	0

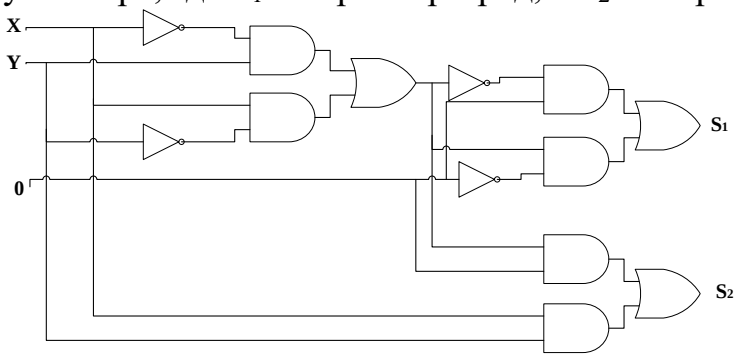
При подсчете S = f(X, Y), где значения для X и Y берутся из первых трех строк, значение совпадает со значениями на калькуляторе. При сложении чисел 0001 и 0001, получается 0010, но четверть сумматор не способен переходить на следующий разряд.

Реализация работы полусумматора.

Полусумматор уже больше похож на калькулятор, но все еще не является полным калькулятором, так как не учитывает переход разряда выше второго.

$$S=(\neg X \wedge Y) \vee (X \wedge \neg Y)$$
$$S1=(\neg S \wedge 0) \vee (S \wedge \neg 0)$$
$$S2=(S \wedge 0) \vee (X \wedge Y)$$

Схема полусумматора, где S_1 – первый разряд, а S_2 – второй.



Программная запись функции, назовем их в программе для дальнейшего использования half_sum_1 и half_sum_2, которые возвращают значения S1 и S2 соответственно, с обращением к функции четверти сумматора – quarter_sum:

```
    def half_sum_1(X, Y):
        return quarter_sum(quarter_sum(X, Y), False)


def half_sum_2(X, Y):
        return (quarter_sum(X, Y) and False) or (X and Y)
```

Подаются значения X и Y и на выходе возвращается S1 и S2.

Таблица логических операций:

X	Y	0	¬X	¬Y	¬X∧Y	X∧¬Y	(¬X∧Y)∨(X∧¬Y)=S
0	0	0	0	1	0	0	0
0	1	0	1	0	1	0	1
1	0	0	0	1	0	1	1
1	1	0	0	0	0	0	0

¬ S	¬0	¬S∧0	S∧¬0	(¬S∧0)∨(S∧¬0)=S1	S∧0	X∧Y	(S∧0)∨(X∧Y)=S2
1	1	0	0	0	0	0	0
0	1	0	1	1	0	0	0
0	1	0	1	1	0	0	0
1	1	0	0	0	0	1	1

В данном случае получаются уже 2 функции – S1 и S2, которые зависят от X и Y. В данном случае сумма чисел 0001 и 0001 уже показывает результат в виде 0 на первом разряде и 1 на втором, то есть 0010.

Реализация работы полного n-битного сумматора.

N-битный сумматор представляет собой последовательное соединение n-ого количества полусумматоров, где для S1 используется в качестве дополнительного разряда 0 и данных значений X1 и Y1, а S2 уже является результатом данных X2 и Y2 и дополнительный разряд равен результату второго разряда работы полусумматора с X1 и Y1.

В качестве примера приведу сумматор для 3-х разрядных чисел.

$$Sn=(\neg Xn\wedge Yn)\vee(Xn\wedge\neg Yn)$$

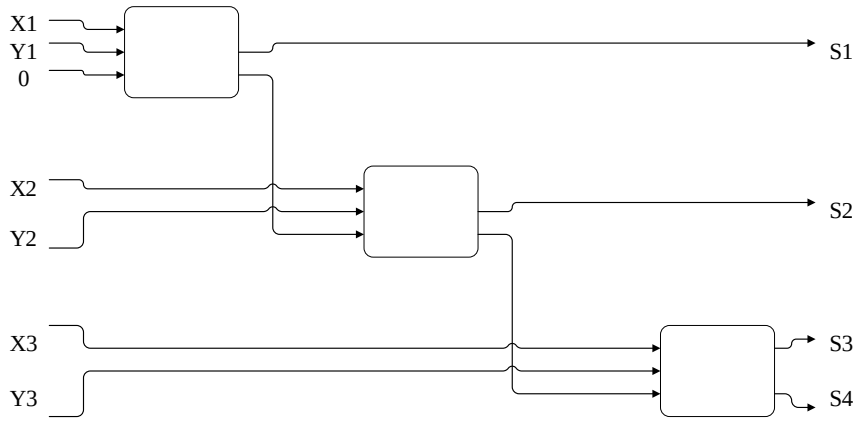
$$S1=(\neg S1\wedge 0)\vee(S1\wedge\neg 0)$$

$$S2=(\neg S2\wedge(S1\wedge 0)\vee(X1\wedge Y1))\vee(S2\wedge\neg(S1\wedge 0)\vee(X1\wedge Y1))$$

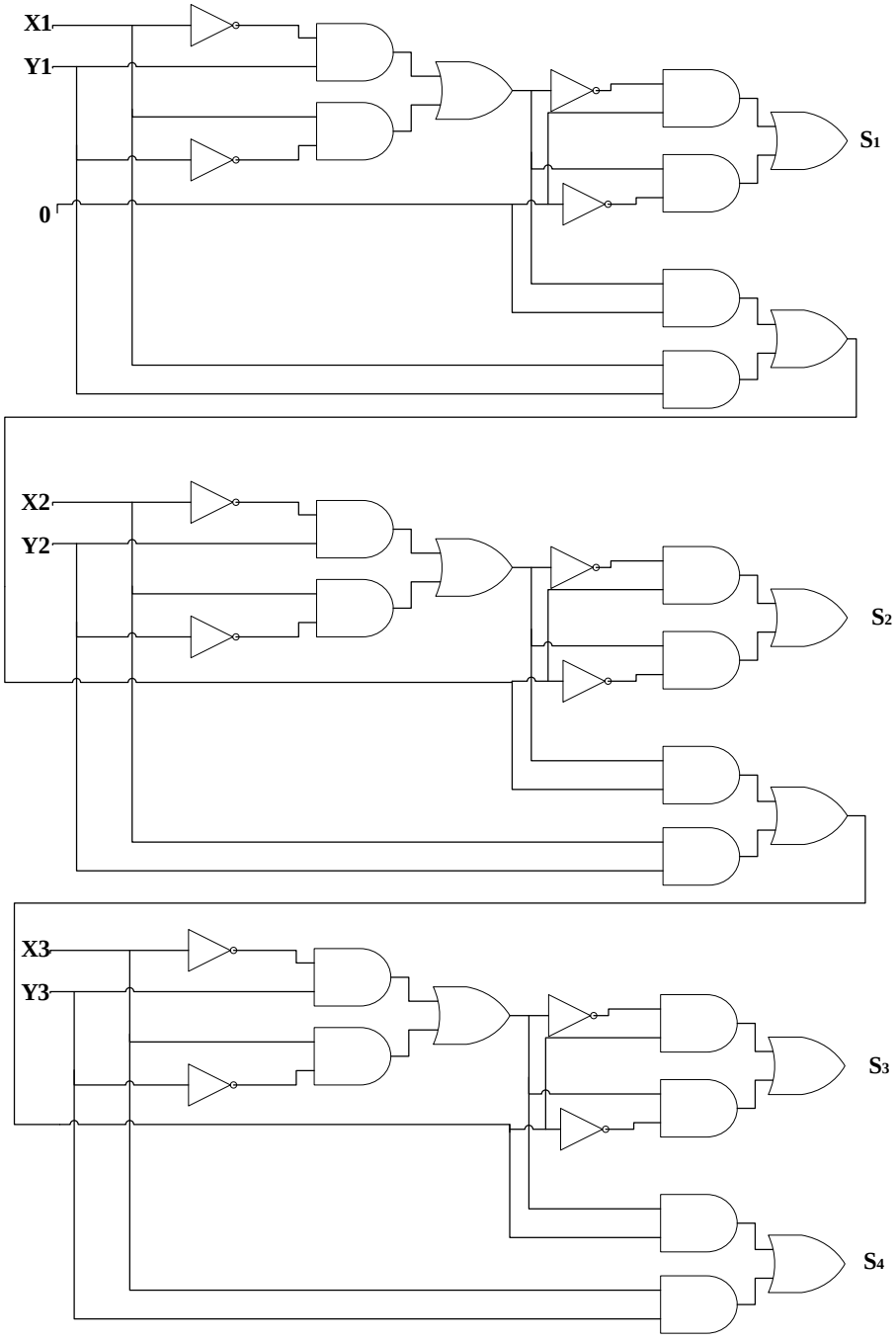
$$S3=(\neg S3\wedge(S2\wedge 0)\vee(X2\wedge Y2))\vee(S3\wedge\neg(S2\wedge 0)\vee(X2\wedge Y2))$$

$$S4=(S3\wedge(S2\wedge 0)\vee(X2\wedge Y2))\vee(X3\wedge Y3)$$

Упрощенная схема 3-битного сумматора, где квадраты являются полусумматорами:



Детальная схема 3-битного сумматора, где числа после X, Y, S обозначают разряд:



Программная запись алгоритма для суммирования двух 3-х разрядных двоичный чисел:

```
        def quarter_sum(X, Y):
            return (not X and Y) or (X and not Y)


def half_sum_1(X, Y, O=False):
            return quarter_sum(quarter_sum(X, Y), O)


def half_sum_2(X, Y, O=False):
            return (quarter_sum(X, Y) and O) or (X and Y)

def sum_3bit(X1, X2, X3, Y1, Y2, Y3):
            S1 = half_sum_1(X1, Y1)
            S2 = half_sum_1(X2, Y2, half_sum_2(X1, Y1))
            S3 = half_sum_1(X3, Y3, half_sum_2(X2, Y2))
            S4 = half_sum_2(X3, Y3, half_sum_2(X2, Y2))
            return [S4, S3, S2, S1]
```

В данном случае мы уже получаем количество функций равное числу на единицу больше разрядности данных изначально чисел. Каждая функция Sn соответствует n-ому разряду числа суммы.

Реализация работы мультиплексора.

Мультиплексор, в отличии от сумматора, совершает умножение первого числа на второе, а не их сложение. Но на самом деле отличий меньше, чем кажется, так как умножение числа m на n равно m+m+…+m(n раз), т.е., разобравшись в устройстве сумматора, каждый сможет реализовать и понять алгоритм работы мультиплексора.

В данной работе нами был разобран принцип действия сумматора. В начале лежит функция $(\neg X \wedge Y) \vee (X \wedge \neg Y)$, а уже на её основе и осложнениях получаются функции, которые позволяют считать числа неограниченные в разрядах.