Практическая реализация

Для практической реализации алгоритма нахождения контрольной суммы был выбран алгоритм CRC8/MAXIM.

Характеристики алгоритма:

1) Производящий полином: $x^8 + x^5 + x^4 + 1$

2) Стартовые данные, то есть значения регистров на момент начала вычислений: 0xFF

3)Флаг (*RefIn*), указывающий на начало и направление вычислений: true

4)Флаг (*RefOut*), определяющий, инвертируется ли порядок битов регистра при входе на элемент XOR: true

5) Число (*XorOut*), с которым складывается по модулю 2 полученный результат: 0x00

6) Значение CRC (*check*) для строки «123456789»: 0xA1

Алгоритм был реализован на языке описания аппаратуры Verilog. Было написано 2 модуля: crc8, реализующий сам алгоритм, и crc_tb, тестирующий модуль crc8.

Сначала идёт объявление модуля crc8 и его входных и выходных сигналов.

Описание входных и выходных сигналов:

1) Входной восьмибитный сигнал data, на который подаются данные

2) Входной однобитный сигнал data_enable, на который подаётся значение, означающие если ещё данные для входа

3) Выходной восьмибитный сигнал crc, через него выводится само значение контрольной суммы

```
module crc8(input[7:0] data,  input data_enable, output reg[7:0] crc );
```

Рис. 1 Объявление модуля crc8

Затем идёт объявление двух переменных: восьмибитного регистра crc_ и счётчика цикла.

```
reg[7:0] crc_ = 8'h00;
integer i;
```

Рис. 2 Переменные

Затем идёт функция fbit_reverse, она нужна для реверса битов.

```
integer i;

function [7:0] fbit_reverse ( input [7:0] data );
integer i;
begin
  for ( i=0; i<8; i=i+1 )
    begin
      fbit_reverse[7-i] = data[i];
    end
  end
endfunction
```

Рис. 3 Функция реверса битов

Затем идёт вычисление crc. Если data_enable равен 1, что означает что у нас есть данные для входа, то мы вычисляем crc. Если данных нет и data_enable равен 0, то мы подаем значение находящиеся в восьмибитном регистре crc_ на вход. И так как флагRefOut равен true, то мы инвертируем порядок бит в регистре crc_ перед тем как подать их на выход.

```
    always @(*) begin
        if(data_enable == 1)
        begin
            crc_  = crc_ ^ fbit_reverse(data);
            for (i = 0; i < 8; i = i + 1)
            begin
                crc_ = crc_ & 8'h80 ? (crc_ << 1) ^ 8'h31 : crc_ << 1;
            end
        end
        else
        begin
            crc = fbit_reverse(crc_);
        end
    end
endmodule
```

Рис. 4 Вычисление crc

Теперь рассмотрим модуль, выполняющий тестирование модуля crc8. Это модуль crc_tb.

Сначала также идёт название модуля. Входных и выходных сигналов у модуля нет. Потом идёт объявление переменных, а также подключение модуля crc8.

```
module crc_tb;
    reg[7:0] data;
    reg clk;
    reg data_enable;
    wire[7:0] crc;
    integer i = 0;

    crc8 test(.data(data), .data_enable(data_enable), .crc(crc));
```

Рис. 5 Начало модуля crc_tb

Далее в блоке initial идёт цикл for, в котором мы подаём на вход data_enable значение 1, а на вход data мы подаём ASCII коды соответствующих символов из строки «123456789». А затем с помощью системной функции $display выводим значение CRC на экран.

```
    initial begin
        for(i = 1; i < 10; i = i + 1)
        begin
            data_enable = 1;
            data = i + 48;
            #10;
        end
        data_enable = 0;
        #10;

        $display("CRC: %X", crc);
    end
endmodule
```

Рис. 6 Конец модуля crc_tb

Выходное значение равно числу 0xA1 что соответствует выходному значению алгоритма CRC8/MAXIM для строки «123456789».

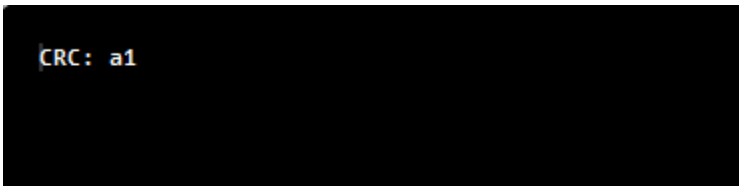

Рис. 7 Выходное значение