mcst_2025

Подготовил Пашенцев Павел Владимирович

Оглавление

Задание

1) Приоритетный шифратор

Описать на Verilog параметризованный модуль, реализующий следующую функциональность: на выходной порт position подаётся значение номера старшего ненулевого разряда данных с входного порта data. При реализации требуется использовать только тип данных wire.

Входные порты: битовый вектор data разрядности DATA_W (задаётся параметром).

Выходные порты: двоичное число position разрядности POS_W (задаётся параметром, по умолчанию может автоматически вычисляться из DATA_W).

*Упрощение: реализуйте указанный модуль для разрядности входа data равной 16, разрядности выхода position равной 4

2) Буфер FIFO

Описать на Verilog параметризованный модуль, реализующий функциональность буфера FIFO; размер буфера задаётся параметром FIFO_SIZE; разрядность данных задаётся параметром DATA_W. При подаче признака записи, данные помещаются в буфер, при подаче признака чтения – удаляются из него по принципу FIFO (First in, First out); признаки записи и чтения могут подаваться одновременно (в одном такте).

Входные порты: однобитный признак write (значимость записи в буфер), данные записи datain разрядности DATA_W, однобитный признак read (значимость чтения из буфера), тактовый сигнал clock, признак сброса reset (активный высокий уровень).

Выходные порты: значимые данные dataout разрядности DATA_W, однобитный признак значимости данных val, однобитный признак заполненности буфера full.

*Упрощение: реализуйте указанный модуль для разрядности портов datain и dataout равной 10, и размера буфера (количества ячеек очереди) равной 6

Выполнение

1) Приоритетный шифратор

В ходе разработке были получены файлы модуля priority_coder.v и тестбенча tb_priority_coder_wire.v.

Был разработан модуль priority_coder.v с учетом требования использования только переменных типа wire. Приоритизация идет на старшую единицу. Если на вход не подается ни одна единица, то на выходе выдаются все единицы.

Был разработан параметризируемый модуль устройства с конфигурацией DATA_W=16 и POS_W=4.

RTL

Была получена RTL-моедель разрабатываемого модуля.

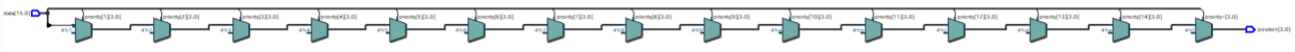


Рисунок 1. RTL-модель устройства

Оценка затраты аппаратных ресурсов

В ходе компиляции проекта была получена информация по потребляемым ресурсам.

Как видно на рисунке 2, для реализации модуля priority_coder_wire было использовано 21 логический элемент и 20 пинов. Можно сделать вывод, что данная схема эффективно использует ресурсы, поскольку количество логических элементов близко к ожидаемому для реализации приоритетного кодировщика на 16 входов (21 ЛЭ достаточно для организации каскада сравнений и мультиплексоров). Количество пинов также соответствует количеству портов ввода-вывода модуля (20 = 16 бит входных данных (DATA_W) + 4 бита позиции (POS_W)). При этом не используются регистры и встроенная память, что характерно для чисто “проводной” (wire-based) реализации без хранения состояния.

Flow Status	Successful - Tue Jun 03 22:11:32 2025
Quartus Prime Version	17.1.0 Build 590 10/25/2017 SJ Lite Edition
Revision Name	priority_coder
Top-level Entity Name	priority_coder_wire
Family	MAX 10
Device	10M50DAF484C7G
Timing Models	Final
Total logic elements	21 / 49,760 (< 1 %)
Total registers	0
Total pins	20 / 360 (6 %)
Total virtual pins	0
Total memory bits	0 / 1,677,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 288 (0 %)
Total PLLs	0 / 4 (0 %)
UFM blocks	0 / 1 (0 %)
ADC blocks	0 / 2 (0 %)

Рисунок 2. Результат компиляции

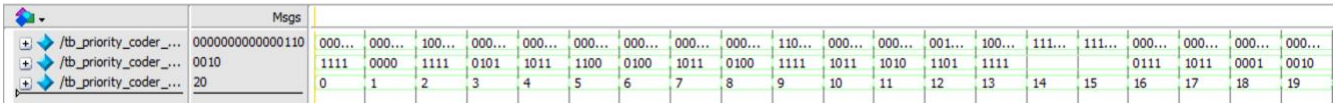
Верификация

Для верикации модуля был разработан тестбенч tb_priority_coder_wire.v. Его симуляция производилась программе Modelsim.

Для упрощения отслеживания возможных ошибок была добавлена переменная expected, с которой происходит сравнение значение вывода. Также выводится корректность прохождения теста Pass/Fail.

Рассматривались различные сценарии: на вход подается ни одно значение, на вход подется one-hot код, на вход подается код с несколькими единицами (проверяется приоритизация).

Как видно на рисунке 4, симуляция произошла успешно.



Рисукнок 3. Симуляция в Modelsim

```
VSIM 102> run -all
# Test 0: data=0000000000000000, position=15, expected=15 -> Pass
# Test 1: data=0000000000000001, position=0, expected=0 -> Pass
# Test 2: data=1000000000000000, position=15, expected=15 -> Pass
# Test 3: data=0000000000100000, position=5, expected=5 -> Pass
# Test 4: data=0000100000000000, position=11, expected=11 -> Pass
# Test 5: data=0001000000000000, position=12, expected=12 -> Pass
# Test 6: data=0000000000010000, position=4, expected=4 -> Pass
# Test 7: data=0000100000000010, position=11, expected=11 -> Pass
# Test 8: data=0000000000010010, position=4, expected=4 -> Pass
# Test 9: data=1100000000001000, position=15, expected=15 -> Pass
# Test 10: data=0000111100000000, position=11, expected=11 -> Pass
# Test 11: data=0000011100000001, position=10, expected=10 -> Pass
# Test 12: data=0011000000000011, position=13, expected=13 -> Pass
# Test 13: data=1000000000001111, position=15, expected=15 -> Pass
# Test 14: data=1111111111111111, position=15, expected=15 -> Pass
# Test 15: data=1111000000000000, position=15, expected=15 -> Pass
# Test 16: data=0000000011111111, position=7, expected=7 -> Pass
# Test 17: data=0000111111111111, position=11, expected=11 -> Pass
# Test 18: data=0000000000000011, position=1, expected=1 -> Pass
# Test 19: data=0000000000000110, position=2, expected=2 -> Pass
```

Рисунок 4. Вывод результата симулции в консоль

2) Буфер FIFO

В ходе разработке были получены файлы модуля fifo.v и тестбенча tb_fifo.v.

Был разработан модуль fifo.v. Реализация основана на хранении регистра данных и выбора значения для вывода с помощью указателя вершины.

Был разработан параметризируемый модуль устройства с конфигурацией DATA_W=10 и FIFO_SIZE=6.

RTL

Была получена RTL-моедель разрабатываемого модуля.

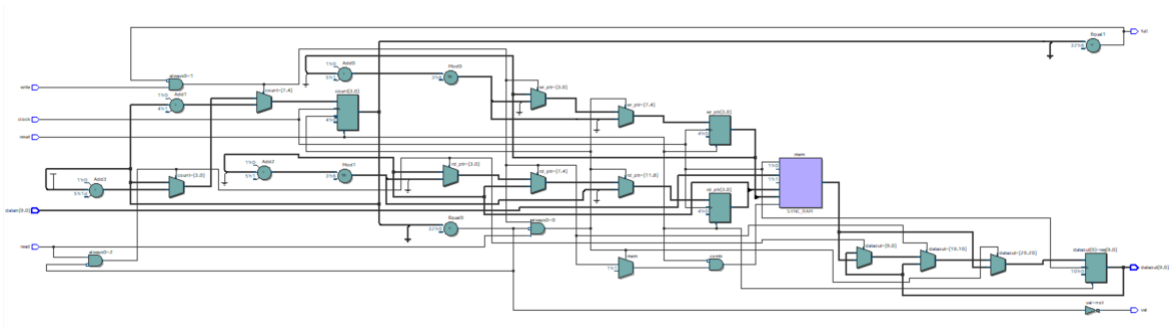


Рисунок 1. RTL-модель устройства

Оценка затраты аппаратных ресурсов

В ходе компиляции проекта была получена информация по потребляемым ресурсам.

Как видно на рисунке 2, для реализации модуля fifo было использовано 123 логических элементов и 26 пинов. Можно сделать вывод, что схема эффективно использует ресурсы, так как число логических элементов соответствует ожидаемому объёму регистрационной логики для хранения данных и реализации управляющих счетчиков (6×10 бит памяти FIFO, указатели и счётчик заполнения).

Количество пинов также сопоставимо с количеством внешних портов модуля: это входы данных (10 бит), выходные данные (10 бит), управляющие сигналы (push, pop, clock, rst_n, full, val) и дополнительные служебные сигналы.

Flow Status	Successful - Tue Jun 03 21:39:28 2025
Quartus Prime Version	17.1.0 Build 590 10/25/2017 SJ Lite Edition
Revision Name	fifo
Top-level Entity Name	fifo
Family	MAX 10
Device	10M50DAF484C7G
Timing Models	Final
Total logic elements	123 / 49,760 (< 1 %)
Total registers	80
Total pins	26 / 360 (7 %)
Total virtual pins	0
Total memory bits	0 / 1,677,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 288 (0 %)
Total PLLs	0 / 4 (0 %)
UFM blocks	0 / 1 (0 %)
ADC blocks	0 / 2 (0 %)

Рисунок 2. Результат компиляции

Также немаловажной характеристикой является максимальная тактовая частота, на которой может работать устройсто памяти. Ведь от этого могут возникнуть проблемы с синхронизацией внешними устройствами. В качестве аппаратной платформы был выбран чип 10M50DAF484C7G (DE10-Lite). В данном случае она составляет 374.95 MHz.

Slow 1200mV 85C Model Fmax Summary

<<Filter>>

	Fmax	Restricted Fmax	Clock Name	Note
1	374.95 MHz	250.0 MHz	clock	lim...te)

Рисунок 3. Максимальная частота

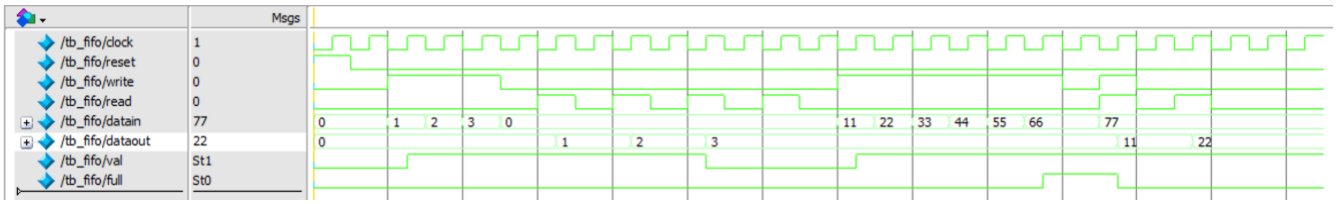
Верификация

Для верикации модуля был разработан тестбенч tb_fifo.v. Его симуляция производилась программе Modelsim.

Рассматривались различные сценарии: производится только запись в пустой буфер, производится только чтение из заполненного буфера, одновременно подаются признаки записи и чтения (проверяется корректность работы при одновременных операциях), выполняется попытка чтения из пустого буфера (проверяется отсутствие изменений и корректность сигнала val), производится запись в полностью заполненный буфер (оценивается работа признака full и блокировка лишних записей), а также последовательная запись и последующее

последовательное чтение всех значений для проверки порядка выдачи данных (FIFO).

Как видно на рисунке 4, симуляция произошла успешно.



Рисукнок 3. Симуляция в Modelsim

```
VSIM 24> run -all
# time=0 | reset=1 | write=0 | read=0 | datain=   0 | dataout=   0 | val=0 | full=0
# time=10000 | reset=0 | write=0 | read=0 | datain=   0 | dataout=   0 | val=0 | full=0
# time=20000 | reset=0 | write=1 | read=0 | datain=   1 | dataout=   0 | val=0 | full=0
# time=25000 | reset=0 | write=1 | read=0 | datain=   1 | dataout=   0 | val=1 | full=0
# time=30000 | reset=0 | write=1 | read=0 | datain=   2 | dataout=   0 | val=1 | full=0
# time=40000 | reset=0 | write=1 | read=0 | datain=   3 | dataout=   0 | val=1 | full=0
# time=50000 | reset=0 | write=0 | read=0 | datain=   0 | dataout=   0 | val=1 | full=0
# time=60000 | reset=0 | write=0 | read=1 | datain=   0 | dataout=   0 | val=1 | full=0
# time=65000 | reset=0 | write=0 | read=1 | datain=   0 | dataout=   1 | val=1 | full=0
# time=70000 | reset=0 | write=0 | read=0 | datain=   0 | dataout=   1 | val=1 | full=0
# time=80000 | reset=0 | write=0 | read=1 | datain=   0 | dataout=   1 | val=1 | full=0
# time=85000 | reset=0 | write=0 | read=1 | datain=   0 | dataout=   2 | val=1 | full=0
# time=90000 | reset=0 | write=0 | read=0 | datain=   0 | dataout=   2 | val=1 | full=0
# time=100000 | reset=0 | write=0 | read=1 | datain=   0 | dataout=   2 | val=1 | full=0
# time=105000 | reset=0 | write=0 | read=1 | datain=   0 | dataout=   3 | val=0 | full=0
# time=110000 | reset=0 | write=0 | read=0 | datain=   0 | dataout=   3 | val=0 | full=0
# time=120000 | reset=0 | write=0 | read=1 | datain=   0 | dataout=   3 | val=0 | full=0
# time=130000 | reset=0 | write=0 | read=0 | datain=   0 | dataout=   3 | val=0 | full=0
# time=140000 | reset=0 | write=1 | read=0 | datain=  11 | dataout=   3 | val=0 | full=0
# time=145000 | reset=0 | write=1 | read=0 | datain=  11 | dataout=   3 | val=1 | full=0
# time=150000 | reset=0 | write=1 | read=0 | datain=  22 | dataout=   3 | val=1 | full=0
# time=160000 | reset=0 | write=1 | read=0 | datain=  33 | dataout=   3 | val=1 | full=0
# time=170000 | reset=0 | write=1 | read=0 | datain=  44 | dataout=   3 | val=1 | full=0
# time=180000 | reset=0 | write=1 | read=0 | datain=  55 | dataout=   3 | val=1 | full=0
# time=190000 | reset=0 | write=1 | read=0 | datain=  66 | dataout=   3 | val=1 | full=0
# time=195000 | reset=0 | write=1 | read=0 | datain=  66 | dataout=   3 | val=1 | full=1
# time=200000 | reset=0 | write=0 | read=0 | datain=  66 | dataout=   3 | val=1 | full=1
# time=210000 | reset=0 | write=1 | read=1 | datain=  77 | dataout=   3 | val=1 | full=1
# time=215000 | reset=0 | write=1 | read=1 | datain=  77 | dataout=  11 | val=1 | full=0
# time=220000 | reset=0 | write=0 | read=0 | datain=  77 | dataout=  11 | val=1 | full=0
# time=230000 | reset=0 | write=0 | read=1 | datain=  77 | dataout=  11 | val=1 | full=0
# time=235000 | reset=0 | write=0 | read=1 | datain=  77 | dataout=  22 | val=1 | full=0
# time=240000 | reset=0 | write=0 | read=0 | datain=  77 | dataout=  22 | val=1 | full=0
```

Рисунок 4. Вывод результата симулции в консоль

Приложение

priority_coder_wire.v

```
module priority_coder_wire
#(
    parameter DATA_W = 16,
    parameter POS_W = 4
)
(
    input wire [DATA_W-1:0] data,
    output wire [POS_W-1:0] position
);

wire [POS_W-1:0] pos [DATA_W-1:0];

genvar i;
generate
    for (i = 0; i < DATA_W; i = i + 1) begin : init_pos
        assign pos[i] = i[POS_W-1:0];
    end
endgenerate


wire [POS_W-1:0] priority [DATA_W-1:0];

assign priority[0] = data[0] ? pos[0] : {POS_W{1'b1}};

generate
    for (i = 1; i < DATA_W; i = i + 1) begin : build_priority
        assign priority[i] = data[i] ? pos[i] : priority[i-1];
    end
endgenerate

assign position = priority[DATA_W-1];

endmodule
```

tb_priority_coder_wire.v

```
`timescale 1ns/1ps

module tb_priority_coder_wire;
 reg  [15:0] data;
 wire [3:0]  position;
 reg [15:0] test_data [0:19];
 reg [3:0] expected [0:19];

 integer i;
 priority_coder_wire dut (
   .data(data),
   .position(position)
 );

 initial begin
   test_data[0]  = 16'b0000_0000_0000_0000; expected[0]  = 15;
   test_data[1]  = 16'b0000_0000_0000_0001; expected[1]  = 0;
   test_data[2]  = 16'b1000_0000_0000_0000; expected[2]  = 15;
   test_data[3]  = 16'b0000_0000_0010_0000; expected[3]  = 5;
   test_data[4]  = 16'b0000_1000_0000_0000; expected[4]  = 11;
   test_data[5]  = 16'b0001_0000_0000_0000; expected[5]  = 12;
   test_data[6]  = 16'b0000_0000_0001_0000; expected[6]  = 4;
   test_data[7]  = 16'b0000_1000_0000_0010; expected[7]  = 11;
   test_data[8]  = 16'b0000_0000_0001_0010; expected[8]  = 4;
   test_data[9]  = 16'b1100_0000_0000_1000; expected[9]  = 15;
   test_data[10] = 16'b0000_1111_0000_0000; expected[10] = 11;
   test_data[11] = 16'b0000_0111_0000_0001; expected[11] = 10;
   test_data[12] = 16'b0011_0000_0000_0011; expected[12] = 13;
   test_data[13] = 16'b1000_0000_0000_1111; expected[13] = 15;
   test_data[14] = 16'b1111_1111_1111_1111; expected[14] = 15;
   test_data[15] = 16'b1111_0000_0000_0000; expected[15] = 15;
   test_data[16] = 16'b0000_0000_1111_1111; expected[16] = 7;
   test_data[17] = 16'b0000_1111_1111_1111; expected[17] = 11;
   test_data[18] = 16'b0000_0000_0000_0011; expected[18] = 1;
   test_data[19] = 16'b0000_0000_0000_0110; expected[19] = 2;

   for (i = 0; i < 20; i = i + 1) begin
```

```
    data = test_data[i];
    #1;
    $display("Test %0d: data=%b, position=%0d, expected=%0d -> %s",
             i, data, position, expected[i],
             (position == expected[i]) ? "Pass" : "FAIL");
  end
  $stop;
 end

endmodule
```

fifo.v

```
module fifo #(
  parameter DATA_W = 10,
  parameter FIFO_SIZE = 6
)(
  input  wire                 clock,
  input  wire                 reset,
  input  wire                 write,
  input  wire                 read,
  input  wire [DATA_W-1:0]    datain,
  output reg  [DATA_W-1:0]    dataout,
  output wire                 val,
  output wire                 full
);

  reg [DATA_W-1:0] mem [0:FIFO_SIZE-1];
  reg [$clog2(FIFO_SIZE):0] rd_ptr;
  reg [$clog2(FIFO_SIZE):0] wr_ptr;
  reg [$clog2(FIFO_SIZE):0] count;

  always @(posedge clock or posedge reset) begin
    if (reset) begin
      rd_ptr  <= 0;
      wr_ptr  <= 0;
      count   <= 0;
      dataout <= 0;
    end else begin
      if (write && !full && read && (count != 0)) begin
```

```
            mem[wr_ptr] <= datain;
            wr_ptr <= (wr_ptr + 1) % FIFO_SIZE;
            dataout <= mem[rd_ptr];
            rd_ptr <= (rd_ptr + 1) % FIFO_SIZE;
         end
         else if (write && !full) begin
            mem[wr_ptr] <= datain;
            wr_ptr <= (wr_ptr + 1) % FIFO_SIZE;
            count <= count + 1;
         end
         else if (read && (count != 0)) begin
            dataout <= mem[rd_ptr];
            rd_ptr <= (rd_ptr + 1) % FIFO_SIZE;
            count <= count - 1;
         end
      end
   end
   assign full = (count == FIFO_SIZE);
   assign val  = (count != 0);
endmodule
```

tb_fifo.v

```
`timescale 1ns/1ps

module tb_fifo;

   parameter DATA_W = 10;
   parameter FIFO_SIZE = 6;

   reg clock;
   reg reset;
   reg write;
   reg read;
   reg [DATA_W-1:0] datain;
   wire [DATA_W-1:0] dataout;
   wire val;
   wire full;

   fifo #(
```

```
    .DATA_W(DATA_W),
    .FIFO_SIZE(FIFO_SIZE)
) dut (
    .clock(clock),
    .reset(reset),
    .write(write),
    .read(read),
    .datain(datain),
    .dataout(dataout),
    .val(val),
    .full(full)
);

initial clock = 0;
always #5 clock = ~clock;

initial begin
    $monitor("time=%0t | reset=%b | write=%b | read=%b | datain=%d | dataout=%d | val=%b | full=%b",
        $time, reset, write, read, datain, dataout, val, full);
end

initial begin
    reset = 1; write = 0; read = 0; datain = 0;
    #10;
    reset = 0;
    #10;

    write = 1; datain = 1; #10;
    write = 1; datain = 2; #10;
    write = 1; datain = 3; #10;
    write = 0; datain = 0;   #10;

    read = 1; #10;
    read = 0; #10;

    read = 1; #10;
    read = 0; #10;
```

```
        read = 1; #10;
        read = 0; #10;

        read = 1; #10;
        read = 0; #10;

        write = 1; datain = 11; #10;
        write = 1; datain = 22; #10;
        write = 1; datain = 33; #10;
        write = 1; datain = 44; #10;
        write = 1; datain = 55; #10;
        write = 1; datain = 66; #10;
        write = 0; #10;

        write = 1; datain = 77; read = 1; #10;
        write = 0; read = 0; #10;

        read = 1; #10;
        read = 0; #10;

        #20;
        $stop;
    end

endmodule
```