# mcst\_2025

Подготовил Пашенцев Павел Владимирович

Оглавление

[Задание 2](#_Toc199882779)

[1) Приоритетный шифратор 2](#_Toc199882780)

[2) Буфер FIFO 2](#_Toc199882781)

[Выполнение 3](#_Toc199882782)

[1) Приоритетный шифратор 3](#_Toc199882783)

[RTL 3](#_Toc199882784)

[Оценка затраты аппаратных ресурсов 3](#_Toc199882785)

[Верификация 4](#_Toc199882786)

[2) Буфер FIFO 6](#_Toc199882787)

[RTL 6](#_Toc199882788)

[Оценка затраты аппаратных ресурсов 6](#_Toc199882789)

[Верификация 7](#_Toc199882790)

[Приложение 9](#_Toc199882791)

[priority\_coder\_wire.v 9](#_Toc199882792)

[tb\_priority\_coder\_wire.v 10](#_Toc199882793)

[fifo.v 11](#_Toc199882794)

[tb\_fifo.v 12](#_Toc199882795)

## 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](file:////priority_coder/priority_coder_wire.v) и тестбенча [tb\_priority\_coder\_wire.v](file:////priority_coder/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.

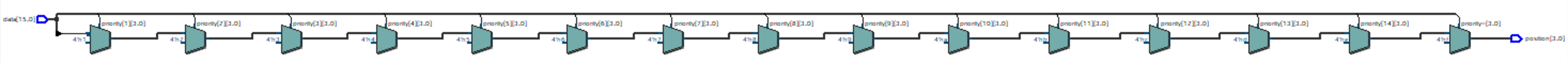


Рисунок 1. RTL-модель устройства

### 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.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 2. Результат компиляции

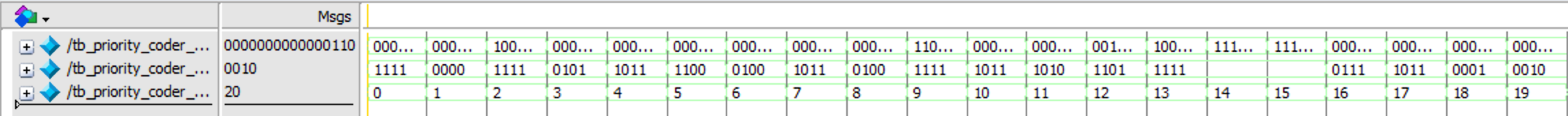
### Верификация

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

Изображение выглядит как текст, снимок экрана, число

Автоматически созданное описание

Рисунок 4. Вывод результата симулции в консоль

## 2) Буфер FIFO

В ходе разработке были получены файлы модуля [fifo.v](file:////fifo/fifo.v) и тестбенча [tb\_fifo.v](file:////fifo/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.

Изображение выглядит как диаграмма, линия, План, снимок экрана

Автоматически созданное описание

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.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

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.

Изображение выглядит как текст, Шрифт, линия, число

Автоматически созданное описание

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.

Изображение выглядит как текст, линия, число, Шрифт

Автоматически созданное описание

Рисукнок 3. Симуляция в Modelsim

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

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