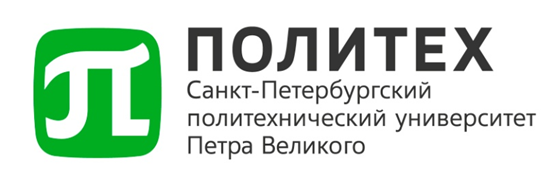
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

***«*САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»**

**Высшая школа программной инженерии**

****

**КУРСОВАЯ РАБОТА**

по дисциплине «**Микроэлектроника, схемотехника**

**и проектирование устройств вычислительной техники**»

на тему

Программная и аппаратная реализация SystemC

**Проекта “Дешифратор 8-256*”***

Студент гр.  Мэн Цзянин

Руководитель Амосов Владимир Владимирович

Санкт-Петербург

1. г

**СОДЕРЖАНИЕ**

[Задание 2](#_Toc41167974)

[Дешифратор 3](#_Toc41167975)

[Разработка программы 4](#_Toc41167976)

[Результаты 8](#_Toc41167977)

[RTL-схема 9](#_Toc41167978)

# Задание

**13. Дешифратор 8-256**

Преобразует двоичный код в позиционный. Разрядность дешифратора задаётся константой. Двоичный код представлен в сокращенной записи (отбрасываются нули перед старшей единицей).

**Входные сигналы:**

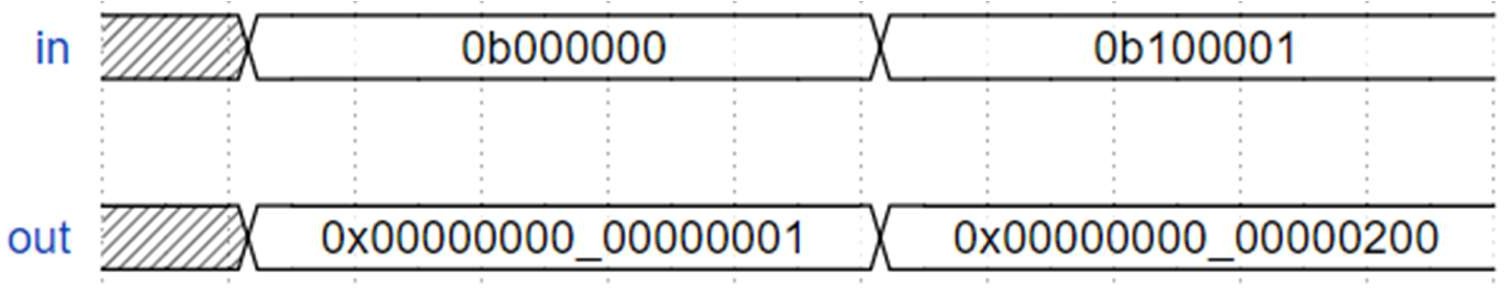
sc\_uint<8> in;

**Выходные сигналы:**

sc\_biguint<256> out;

**Логика:**

комбинационная

**Пример временной диаграммы:**

# Дешифратор

Дешифратор - Это комбинационное устройство, позволяющее распознавать числа, представленные позиционным n-разрядным кодом.

Дешифратор (декодер) в цифровой электронике — комбинационная схема, преобразующая n-разрядный двоичный, троичный или k ‑ ичный код в kn ‑ ичный одноединичный код, где k — основание системы счисления.

# Разработка программы

* **decoder.h**

1. #include <iostream>
2. #include "systemc.h"
4. // decoder module
5. SC\_MODULE(decoder)
6. {
7. sc\_in<**bool**> enable;
8. sc\_in<sc\_uint<8>> input;
9. sc\_out<sc\_biguint<256>> output;
11. **void** proc\_decoder();
13. // Constructor function
14. SC\_CTOR(decoder)
15. {
16. SC\_METHOD(proc\_decoder);
17. sensitive << input << enable;
18. }
20. };
22. // driver
23. SC\_MODULE(driver)
24. {
25. sc\_out<**bool**> d\_enable;
26. sc\_out<sc\_uint<8>> d\_input;
28. **void** proc\_driver();
30. SC\_CTOR(driver)
31. {
32. SC\_THREAD(proc\_driver);
33. }
35. };

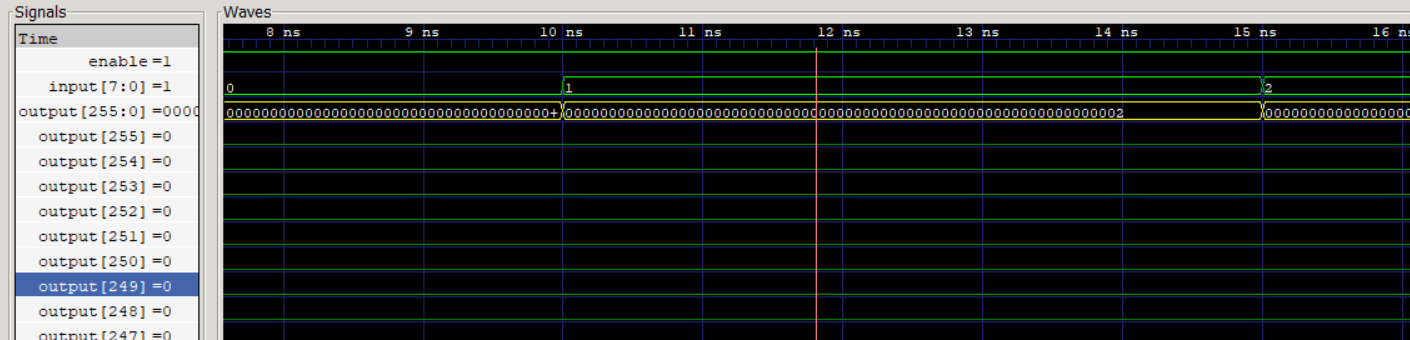
* **decoder.cpp**

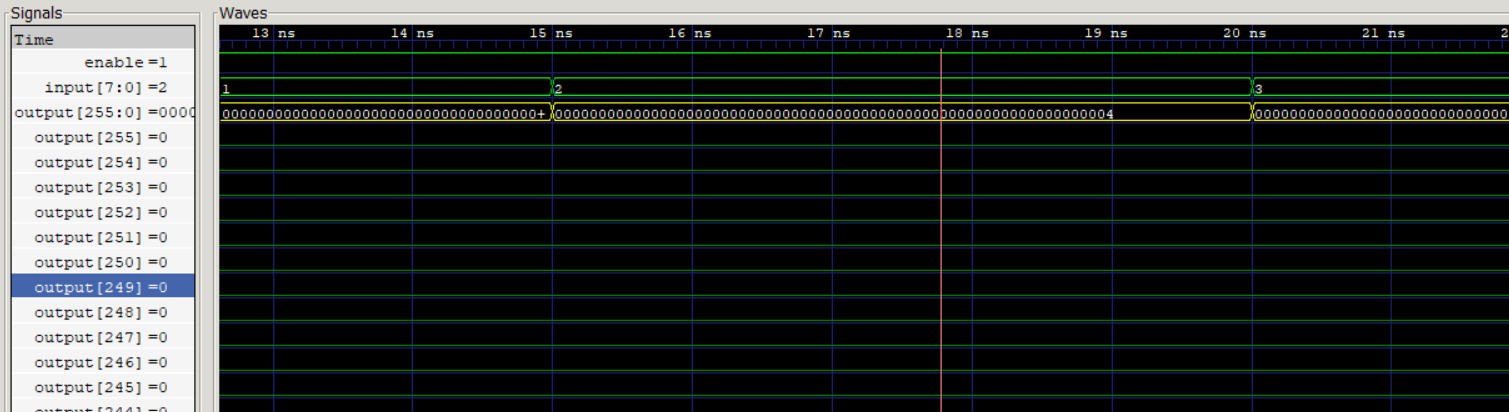
1. #include "decoder.h"
3. **void** decoder::proc\_decoder()
4. {
5. **if** (enable.read())
6. {
7. sc\_uint<8> value\_read = input.read();
9. **for** (**int** i = 0; i <= 256; i++)
10. {
11. **if** (i == value\_read)
12. {
13. output.write(pow(2, i)); // 2 的 i 次方
14. }
15. }
17. }
18. **else**
19. {
20. output.write(0); // if the enable is zero
21. }
22. }
24. **void** driver::proc\_driver()
25. {
26. sc\_uint<8> pattern;
27. pattern = 0;
29. **while** (**true**)
30. {
31. // Test-bench
32. pattern = 0;
33. d\_enable = **false**;
34. d\_input = pattern;
36. wait(5, SC\_NS);
38. **for** (**int** i = 0; i < 256; i++)
39. {
40. pattern = i;
41. d\_enable = **true**;
42. d\_input = pattern;
43. wait(5, SC\_NS);
44. }
46. pattern = 256;
47. d\_enable = **false**;
48. d\_input = pattern;
50. wait(5, SC\_NS);
52. sc\_stop(); // Stop the simulation
53. }
54. }

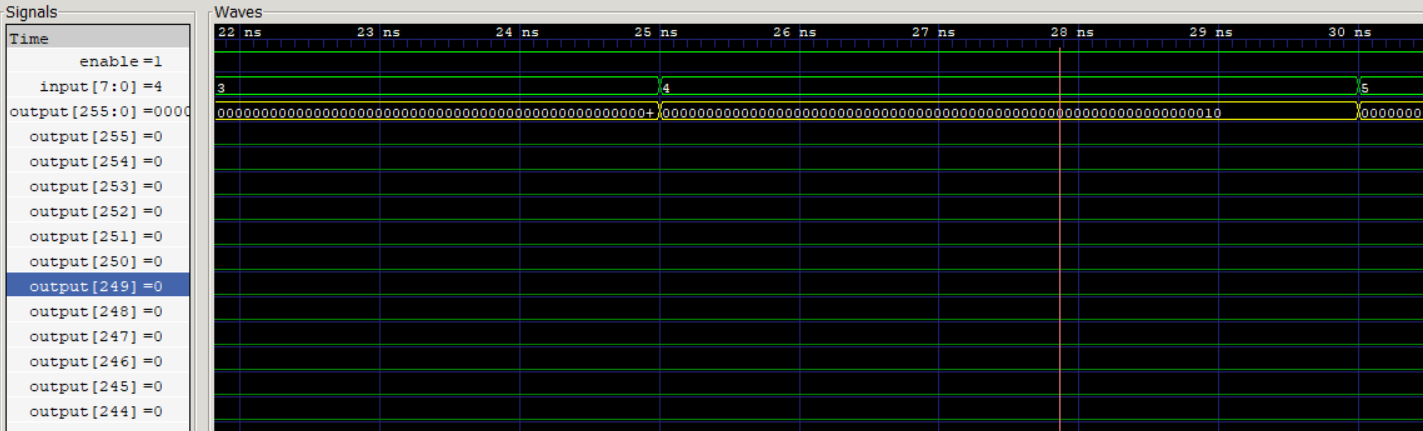
* **main.cpp**

1. #include "decoder.h"
3. **int** sc\_main(**int** argc, **char**\* argv[])
4. {
5. sc\_signal<**bool**> t\_enable;
6. sc\_signal<sc\_uint<8>> t\_input; // int 整形变量
7. sc\_signal<sc\_biguint<256>> t\_output; // 当整数值超过int数据范围时才可以使用
9. // Driver
10. driver DRIVER("DRIVER");
11. DRIVER.d\_enable(t\_enable);
12. DRIVER.d\_input(t\_input);
14. // main module
15. decoder DECODER("decoder");
16. DECODER.enable(t\_enable);
17. DECODER.input(t\_input);
18. DECODER.output(t\_output);
20. // Output signals into .vcd file
21. sc\_trace\_file\* trace = sc\_create\_vcd\_trace\_file("decoder");
23. sc\_trace(trace, t\_enable, "enable");
24. sc\_trace(trace, t\_input, "input");
25. sc\_trace(trace, t\_output, "output");
27. sc\_start(20000, SC\_NS);
29. sc\_close\_vcd\_trace\_file(trace);
31. **return** 0;
32. }

# **Результаты**









# RTL-схема

С помощью инструментов Quartus II 5.0 были получены RTL-схема и техническая схема.

