Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Cанкт-Петербургский государственный университет телекоммуникаций

им. проф. М.А. Бонч-Бруевича»

**Курсовая работа**

Проектирование конечного автомата с

реализацией в ПЛИС

по дисциплине «Программное проектирование элементов

вычислительных систем»

Вариант №8

«Конечный автомат, управляющий работой подъемного крана»

Студенты:

Лапин Роман Владимирович

Хрусталев Никита Эдуардович

Чирков Алексей Дмитриевич

Курс 2 Группа ИКВТ-11 Преподаватель:

Неелова Ольга Леонидовна

Санкт-Петербург

2023

# ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 3](#_bookmark0)

[ЗАДАНИЕ НА КУРСОВОЕ ПРОЕКТИРОВАНИЕ 4](#_bookmark1)

[Общее задание на курсовое проектирование 4](#_bookmark2)

[Вариант задания на курсовое проектирование 4](#_bookmark3)

[Диаграмма состояний 5](#_bookmark4)

[Работа устройства 5](#_bookmark5)

[Код программы 6](#_bookmark6)

[СПИСОК ЛИТЕРАТУРЫ 10](#_bookmark7)

# ВВЕДЕНИЕ

**Конечный автомат** – модель вычислительного устройства с фиксированным и конечным объемом памяти, которое читает и обрабатывает цепочку входных символов, принадлежащих некоторому конечному множеству. Конечные автоматы различают в зависимости от того, какой результат они дают на выходе.

**Абстрактный автомат** является математической моделью дискретного устройства и описывается шестикомпонентным набором:

𝑆 = (𝑠𝑡𝑎𝑡𝑒𝑠, 𝑖𝑛𝑝𝑢𝑡𝑠, 𝑜𝑢𝑡𝑝𝑢𝑡𝑠, 𝑓𝑡𝑟𝑎𝑛𝑠𝑖𝑡𝑖𝑜𝑛, 𝑓𝑜𝑢𝑡𝑝𝑢𝑡, 𝑖𝑛𝑖𝑡𝑖𝑎𝑙\_𝑠𝑡𝑎𝑡𝑒)

* 𝑠𝑡𝑎𝑡𝑒𝑠 — множество состояний.
* 𝑖𝑛𝑝𝑢𝑡𝑠 — множество входных сигналов.
* 𝑜𝑢𝑡𝑝𝑢𝑡𝑠 — множество выходных сигналов.
* 𝑓𝑡𝑟𝑎𝑛𝑠𝑖𝑡𝑖𝑜𝑛 — функция переходов.
* 𝑓𝑜𝑢𝑡𝑝𝑢𝑡 — функция выходов.
* 𝑖𝑛𝑖𝑡𝑖𝑎𝑙\_𝑠𝑡𝑎𝑡𝑒 — начальное состояние.

**Автомат Мили** (автомат первого рода) — конечный автомат, выходная последовательность которого зависит от состояния автомата и входных сигналов.

{𝑠𝑡𝑎𝑡𝑒(𝑡 + 1) = 𝑓𝑡𝑟𝑎𝑛𝑠𝑖𝑡𝑖𝑜𝑛(𝑠𝑡𝑎𝑡𝑒(𝑡), 𝑖𝑛𝑝𝑢𝑡(𝑡))

𝑜𝑢𝑡𝑝𝑢𝑡(𝑡) = 𝑓𝑜𝑢𝑡𝑝𝑢𝑡(𝑠𝑡𝑎𝑡𝑒(𝑡), 𝑖𝑛𝑝𝑢𝑡(𝑡))

Где 𝑠𝑡𝑎𝑡𝑒(𝑡) — текущее состояние, 𝑠𝑡𝑎𝑡𝑒(𝑡 + 1) — следующее состояние,

𝑓𝑡𝑟𝑎𝑛𝑠𝑖𝑡𝑖𝑜𝑛 — функция переходов, 𝑖𝑛𝑝𝑢𝑡(𝑡) — входной сигнал текущего состояния,

𝑓𝑜𝑢𝑡𝑝𝑢𝑡 — функция выходов, 𝑜𝑢𝑡𝑝𝑢𝑡(𝑡) — выходной сигнал текущего состояния.

**Автомат Мура** (автомат второго рода) — конечный автомат, выходная последовательность которого зависит только от состояния автомата, и не зависит напрямую от входных сигналов.

{𝑠𝑡𝑎𝑡𝑒(𝑡 + 1) = 𝑓𝑡𝑟𝑎𝑛𝑠𝑖𝑡𝑖𝑜𝑛(𝑠𝑡𝑎𝑡𝑒(𝑡), 𝑖𝑛𝑝𝑢𝑡(𝑡))

𝑜𝑢𝑡𝑝𝑢𝑡(𝑡) = 𝑓𝑜𝑢𝑡𝑝𝑢𝑡 (𝑠𝑡𝑎𝑡𝑒(𝑡))

Где 𝑠𝑡𝑎𝑡𝑒(𝑡) — текущее состояние, 𝑠𝑡𝑎𝑡𝑒(𝑡 + 1) — следующее состояние,

𝑓𝑡𝑟𝑎𝑛𝑠𝑖𝑡𝑖𝑜𝑛 — функция переходов, 𝑖𝑛𝑝𝑢𝑡(𝑡) — входной сигнал текущего состояния,

𝑓𝑜𝑢𝑡𝑝𝑢𝑡 — функция выходов, 𝑜𝑢𝑡𝑝𝑢𝑡(𝑡) — выходной сигнал текущего состояния.

Причем справедлива эквивалентность автоматов Мили и Мура: для каждого автомата Мили может быть построен эквивалентный ему автомат Мура, и обратно — для каждого автомата Мура может быть построен эквивалентный ему автомат Мили.

**Автоматы Мили и Мура** широко применяются при проектировании цифровых устройств на основе программируемых логических интегральных схем (ПЛИС).

Основное преимущество использования автомата Мили заключается в возможности реакции автомата в течение текущего такта, что обусловлено зависимостью текущей выходной комбинации от текущей входной комбинации.

Наличие минимальной выходной задержки, связанной с переключением выходного регистра, отсутствие нестабильности переходного процесса на выходе автомата, отсутствие сквозного распространения сигнала через комбинационную схему от входа до выхода автомата, простота описания на языках описания аппаратуры HDL делает автомат Мура практически незаменимым.

# ЗАДАНИЕ НА КУРСОВОЕ ПРОЕКТИРОВАНИЕ

# Общее задание на курсовое проектирование

Необходимо синтезировать конечный автомат с произвольной сменой состояний, каждое из которых отражает некоторое событие. Возможность перехода из одного состояния в другое зависит от сигнала *x*, который формируется с помощью счетчика тактов. В некоторых заданиях также предусматривается формирование дополнительного управления от внешних источников.

В синтез входит:

1. Построение временной диаграммы работы счетчика тактов с указанием интервалов времени формирования управляющего сигнала *x*;
2. Программа на Verilog HDL для реализации автомата на макете в FPGA CycloneV;
3. Диаграмма состояний автомата.

Таблица переходов отражает ход смены состояний с учетом времени удержания предшествующего состояния в тактах.

Таблица состояний отражает событие, соответствующее полученному состоянию, и отображение состояния на выходной шине. Состояние выходной шины при реализации выводится на сегментные индикаторы.

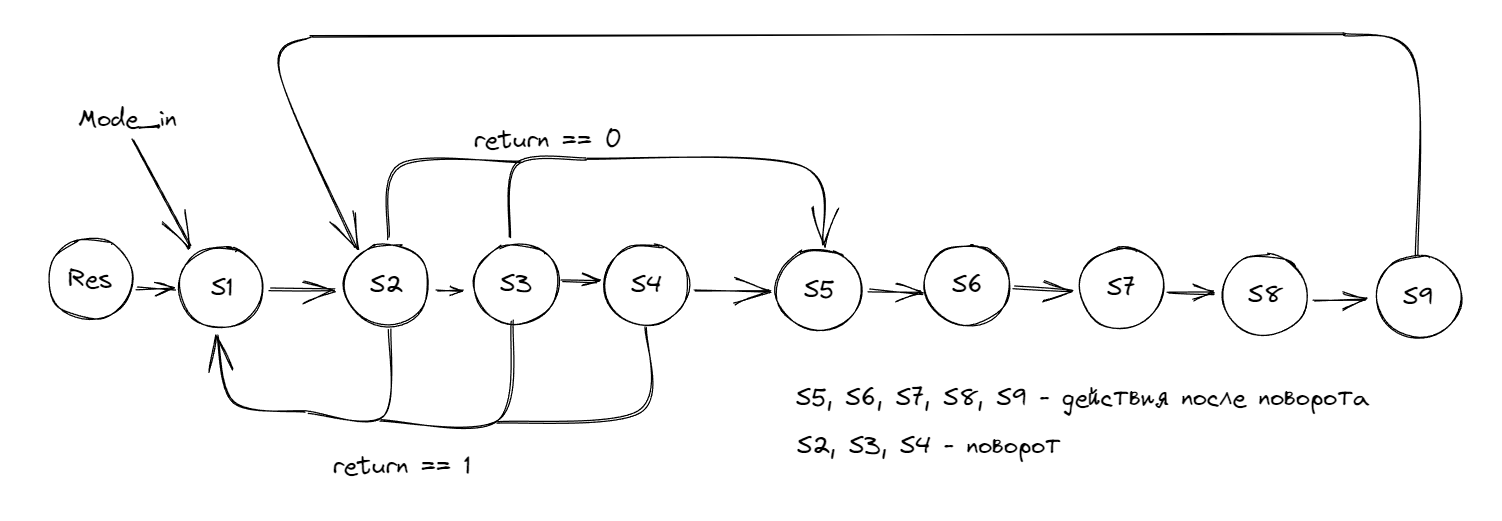
# Вариант задания на курсовое проектирование

Выбранный вариант: «Автомат, управляющий работой подъемного крана».

Состояния автомата:

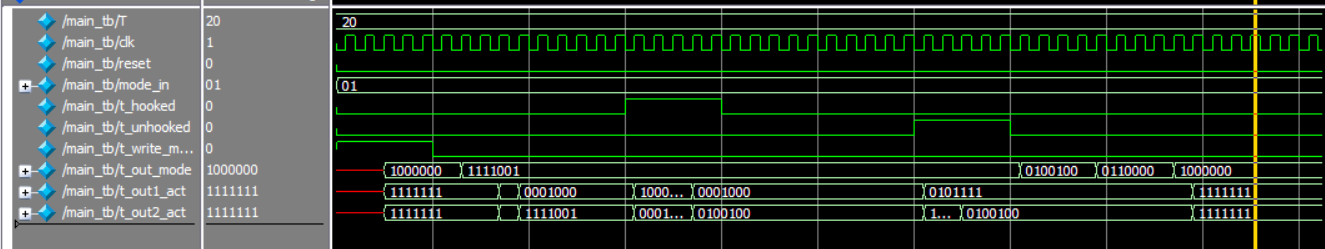
* (Res) — начальное состояние.
* (𝑆1) — считывание установленного режима в зависимости от угла поворота.
* (𝑆2 – S4) — Ожидание окончания поворота.
* S5 - движение вниз
* S6 – ожидание сигнала о закреплении груза
* S7 – движение движение вверх до высоты, определенной выставленной в начальном условии константой
* S8 – ожидание сигнала об отцеплении груза
* S9 – движение вниз до исходного положения после поворота стрелы

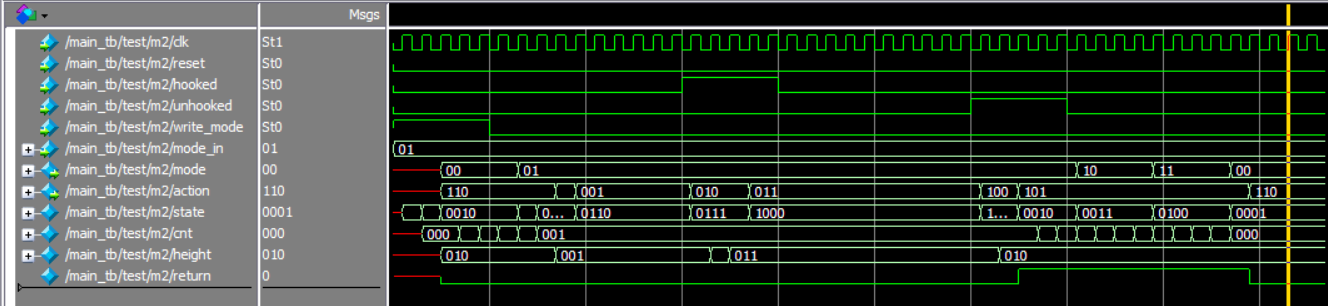
# Диаграмма состояний

На рисунке 1 представлена диаграмма состояний конечного автомата.

# Работа устройства

Посмотрим на следующий рисунок.

Рисунок 1. Диаграмма состояний



# Параметры:

* **T** — период синхроимпульса.

Рисунок 2. Пример работы устройства (тестбенч-файл)

# Источники сигнала:

* **clk** — синхроимпульс.
* **res** — синхронный сброс счетчика cnt при 1.
* **reset** — асинхронный сброс счетчика cnt при 0.
* **state** — текущее состояние конечного автомата.
* **cnt** — счетчик.

# Последовательный вывод на индикаторах:

# Код программы

# count\_div2.v:

# module count\_div2

# #(parameter N=29,M=29'd25000000)

# (input wire clk,

# output reg sync = 0,

# output wire [N-1:0]q);

# reg [N-1:0]cnt=0;

# wire [N-1:0]cnt\_next;

# assign q = cnt;

# assign cnt\_next = cnt + 1'b1;

# always@(posedge clk) begin

# cnt <= cnt\_next;

# sync <= sync;

# if (cnt == M-1) begin

# cnt <= 0;

# sync <= ~sync;

# end

# end

# endmodule

# coder\_action.v:

# module coder\_action

# #(parameter dn = 3'b000, A1 = 3'b001, up = 3'b010, A2 = 3'b011,

# r1 = 3'b100, r2 = 3'b101, nothing = 3'b110)

# (input wire [2:0] data,

# output wire [6:0] seg1\_action,

# output wire [6:0] seg2\_action);

# reg [6:0]code1;

# reg [6:0]code2;

# assign seg1\_action=code1;

# assign seg2\_action=code2;

# always @\*

# begin

# case(data)

# dn:

# begin

# code1 = 7'b0100001;

# code2 = 7'b1101011;

# end

# A1:

# begin

# code1 = 7'b0001000;

# code2 = 7'b1111001;

# end

# up:

# begin

# code1 = 7'b1000001;

# code2 = 7'b0001100;

# end

# A2:

# begin

# code1 = 7'b0001000;

# code2 = 7'b0100100;

# end

# r1:

# begin

# code1 = 7'b0101111;

# code2 = 7'b1111001;

# end

# r2:

# begin

# code1 = 7'b0101111;

# code2 = 7'b0100100;

# end

# nothing:

# begin

# code1 = 7'b1111111;

# code2 = 7'b1111111;

# end

# endcase

# end

# endmodule

# coder\_mode.v:

# module coder\_mode

# (input wire [1:0] data,

# output wire [6:0] seg\_mode);

# reg [6:0]code;

# assign seg\_mode=code;

# always @\*

# case(data)

# 2'b00: code = 7'b1000000; // 0: 0

# 2'b01: code = 7'b1111001; // 1: 90

# 2'b10: code = 7'b0100100; // 2: 180

# 2'b11: code = 7'b0110000; // 3: -90

# endcase

# endmodule

# moore.v

# module moore

# #(parameter up\_to = 3'd3, down\_to = 3'd1, start\_height = 3'd2, turn\_time = 3'd3,

# dn = 3'b000, A1 = 3'b001, up = 3'b010, A2 = 3'b011,

# r1 = 3'b100, r2 = 3'b101, nothing = 3'b110)

# (input clk, reset, hooked, unhooked, write\_mode,

# [1:0]mode\_in, // required angle (mode)

# output reg [1:0] mode,

# [2:0] action);

# reg [3:0] state;

# reg [2:0] cnt;

# reg [2:0] height;

# reg return;

# parameter Res = 0, S1 = 1, S2 = 2, S3 = 3, S4 = 4, S5 = 5, S6 = 6, S7 = 7, S8 = 8, S9 = 9;

# always @(posedge clk or posedge reset) begin

# if (reset)

# state <= Res;

# else begin

# case(state)

# Res:

# state <= S1;

# S1:

# begin

# if (return) begin

# action <= nothing;

# return <= 1'b0;

# end

# if (write\_mode) begin

# mode <= 2'd0;

# action <= nothing;

# return <= 1'b0;

# height <= start\_height;

# state <= S2;

# end

# end

# S2:

# begin

# if (!return) begin

# if(cnt == turn\_time)

# state <= S3;

# if (mode == mode\_in) begin

# state <= S5;

# end

# end

# else begin

# if(cnt == turn\_time)

# state <= S3;

# if (mode == 3'd0)

# state <= S1;

# end

# end

# S3:

# begin

# if (!return) begin

# if(cnt == turn\_time)

# state <= S4;

# if (mode == mode\_in) begin

# state <= S5;

# end

# end

# else begin

# if(cnt == turn\_time)

# state <= S4;

# if (mode == 3'd0)

# state <= S1;

# end

# end

# S4:

# begin

# if (!return) begin

# if(cnt == turn\_time || mode == mode\_in) begin

# state <= S5;

# end

# end

# else begin

# if(cnt == turn\_time || mode == 3'd0) begin

# state <= S1;

# end

# end

# end

# S5:

# if (height == down\_to) begin

# action <= A1;

# state <= S6;

# end

# S6:

# if (hooked == 1'b1) begin

# action <= up;

# state <= S7;

# end

# S7:

# if (height == up\_to) begin

# action <= A2;

# state <= S8;

# end

# S8:

# if (unhooked == 1'b1) begin

# action <= r1;

# state <= S9;

# end

# S9:

# if (height == start\_height) begin

# return <= 1'b1;

# action <= r2;

# state <= S2;

# end

# default:

# state <= Res;

# endcase

# end

# end

# always @ (posedge clk) begin

# case(state)

# Res:

# begin

# cnt <= 3'd0;

# end

# S2:

# if (cnt == 3'd3) begin

# cnt <= 3'd0;

# mode <= mode + 3'd1;

# end

# else begin

# cnt <= cnt + 3'd1;

# end

# S3:

# if (cnt == 3'd3) begin

# cnt <= 3'd0;

# mode <= mode + 3'd1;

# end

# else begin

# cnt <= cnt + 3'd1;

# end

# S4:

# if (cnt == 3'd3) begin

# cnt <= 3'd0;

# mode <= mode + 3'd1;

# end

# else begin

# cnt <= cnt + 3'd1;

# end

# S5:

# begin

# if (action != dn)

# action <= dn;

# if (height != down\_to) begin

# height <= height - 3'd1;

# end

# end

# S6:

# if (action != A1)

# action <= A1;

# S7:

# begin

# if (action != up)

# action <= up;

# if (height != up\_to) begin

# height <= height + 3'd1;

# end

# end

# S8:

# if (action != A2)

# action <= A2;

# S9:

# begin

# if (action != r1)

# action <= r1;

# if (height > start\_height) begin

# height <= height - 3'd1;

# end

# else if (height < start\_height) begin

# height <= height + 3'd1;

# end

# end

# endcase

# end

# endmodule

# main.v

# module main

# (input wire clk, reset, hooked, unhooked, write\_mode,

# [1:0]mode\_in,

# output wire [6:0]out\_mode,

# [6:0] out1\_action,

# [6:0] out2\_action);

# wire [1:0] mode;

# wire [2:0] action;

# wire sdiv;

# count\_div2 m1(.clk(clk) ,.sync(sdiv));

# moore m2(.clk(sdiv), .reset(reset), .mode(mode), .mode\_in(mode\_in), .hooked(hooked), .unhooked(unhooked), .action(action), .write\_mode(write\_mode));

# coder\_mode m3(.data(mode), .seg\_mode(out\_mode));

# coder\_action m4(.data(action), .seg1\_action(out1\_action), .seg2\_action(out2\_action));

# endmodule

# main\_tb.v

# `timescale 1ns/10ps

# module main\_tb;

# localparam T=20;

# reg clk;

# reg reset;

# reg [1:0]mode\_in;

# reg t\_hooked;

# reg t\_unhooked;

# reg t\_write\_mode;

# wire [6:0]t\_out\_mode;

# wire [6:0]t\_out1\_act;

# wire [6:0]t\_out2\_act;

# main test(.clk(clk) ,.reset(reset) ,.out\_mode(t\_out\_mode) ,.mode\_in(mode\_in), .out1\_action(t\_out1\_act),

# .out2\_action(t\_out2\_act), .hooked(t\_hooked), .unhooked(t\_unhooked), .write\_mode(t\_write\_mode));

# always begin

# clk = 1'b0;

# #(T/2);

# clk = 1'b1;

# #(T/2);

# end

# initial begin

# mode\_in = 2'd1;

# reset = 1'b0;

# t\_hooked = 1'b0;

# t\_unhooked = 1'b0;

# t\_write\_mode = 1'b0;

# 

# t\_write\_mode = 1'b1;

# #(T\*5);

# t\_write\_mode = 1'b0;

# #(T\*10);

# t\_hooked = 1'b1;

# #(T\*5);

# t\_hooked = 1'b0;

# #(T\*10);

# t\_unhooked = 1'b1;

# #(T\*5);

# t\_unhooked = 1'b0;

# #(T\*10);

# #(T\*10000);

# $stop;

# end

# endmodule

# СПИСОК ЛИТЕРАТУРЫ

1. **Неелова О. Л. Лекции по дисциплине «Программное проектирование элементов вычислительных систем»**.
2. **Автоматы Мили и Мура** [Электронный ресурс]. – Режим доступа: <https://neerc.ifmo.ru/wiki/index.php?title=Автоматы_Мура_и_Мили>, свободный – (29.05.2020).
3. **Угрюмов Е. П. Цифровая схемотехника**: учеб. пособие для вузов. — 3-е изд., перераб. и доп. — СПб.: БХВ-Петербург, 2010. — 816 с.: ил.