**2023 - 2024 学年第 1 学期**

**《数字电子技术B》 课程设计报告**

**设计题目： 出租车计价器**

**成员分工表**

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **学号** | **姓名** | **分工** |
| 1 | 2022112566 | 倪阳 | 负责编写taximete.v到taximete5.v模块的代码设计。实现计价功能的逻辑和计算。确保模块之间的信号传递和数据处理正确无误。 |
| 2 | 2022112560 | 许铮 | 负责编写tset\_taximete.v等仿真测试代码。编写仿真测试用例，涵盖各种输入情况和边界情况。运行仿真工具并分析仿真结果，确保代码能够正确工作。与成员A协调，调试和优化代码设计。 |

**2023年12月**

**摘要**

本设计旨在开发一个出租车计价器，实现准确的里程计费和时间计费功能，同时包括显示和控制模块。通过模块化设计和Quartus编程，实现了对出租车计价器的功能划分和代码开发。在完成代码设计后，通过使用Quartus的仿真工具对代码进行正确性分析和验证，确保系统能够按照预期工作。通过该设计，可以实现精准的出租车计费，并为用户提供方便的计费信息显示。

**摘要内容**

本设计实现了一个功能完备的出租车计价器。通过将整个系统划分为计价模块、显示模块和控制模块，实现了准确的里程计费和时间计费，并将计费结果通过数码管进行显示。在模块化设计的基础上，分配了不同的任务给不同的成员，确保了代码开发和测试的高效进行。通过使用Quartus的仿真工具，对代码进行仿真测试，并分析了结果的正确性和准确性。该设计为出租车计价器的开发提供了一个可行的方案，能够满足计费、显示和控制的需求，并提供了一个稳定可靠的计价器系统。

目录

[**一、设计任务**](#_Toc533240467) 3

[1. 基本要求功能](#_Toc533240468) 3

[2. 拓展功能](#_Toc533240469) 3

[3. 完成设计所用时间](#_Toc533240470) 3

[**二、设计原理与仿真验证**](#_Toc533240471) 4

[1.设计原理](#_Toc533240472) 4

内容：设计原理、总体方案，顶层框图

[2.各模块设计与仿真分析](#_Toc533240474) 4

内容：各模块设计原理、状态机、VHDL源代码、RTL图、仿真波形与分析等

[3.总体仿真与分析](#_Toc533240482) 26

内容：仿真波形、正确性分析等

[**三、心得体会**](#_Toc533240490) 28

[**四、参考文献**](#_Toc533240491) 28

[**五、附件**](#_Toc533240492)[**Quartus 工程文件夹**](#_Toc533240493) 28

1. **设计任务**
2. 基本要求功能
3. 拓展功能

主体功能为出租车自动计费器，按照题目要求，进行各种费用的计费叠加。

在这之下实现的功能有：

1. 显示汽车行驶里程（用四位数字显示，单位为km。计程范围为0~99km，计程分辨率为1km。）
2. 显示等候时间（用两位数字显示分钟，单位为min。计时范围为0~59min，计时分辨率为1min。）

同时，用直流电机的转动模拟出租车行驶的状态，通过计算直流电机的转速和圈数得到里程数。

1. 完成设计所用时间

2023.12.11~2023.12.19

1. **设计原理与仿真验证**
2. 设计原理

（1）、总体原理：出租车计价器的设计旨在根据里程和时间实现准确的计费，同时需要包括控制逻辑、数码管显示等功能。

计价原理：计价模块根据里程和时间进行计费，输出计费结果。

显示原理：数码管显示模块接收计费结果并进行数码管显示。

控制原理：控制模块负责整体系统的控制和时序生成。

（2）、总体方案：

计价器功能：实现里程计费和时间计费，能够实时显示计费结果。

硬件平台：基于FPGA实现，通过Verilog语言描述数字电路。

模块化设计：将整个系统划分为功能模块，如计价模块、显示模块、控制模块等，便于分工合作。

（3）、顶层框图：

计价模块：根据里程和时间进行计费，输出计费结果。

显示模块：接收计费结果并进行数码管显示。

控制模块：负责整体系统的控制和时序生成。

1. 各模块设计与仿真分析

源代码：

taximeter.v：

module taximeter (

clk,rst,motor,op,mil,dis,timee,sec,cost,seg,codeout,cnt

);

input clk;

input rst;

input motor;

input [1:0] op;

output [7:0] mil;

output [17:0] dis;

output [6:0] timee;

output [6:0] sec;

output [10:0] cost;

output [7:0] seg;

output [7:0] codeout;

output [2:0] cnt;

// wire clk1000;

// taximeter0 i0(clk,clk1000);

taximeter1 i1(clk,rst,motor,mil,dis);

taximeter2 i2(clk,rst,motor,timee,sec);

taximeter3 i3(mil,timee,cost);

taximeter4 i4(clk,op,mil,timee,cost,seg,codeout,cnt);

endmodule

设计原理：

这是一个计程车收费计费模块的设计原理，该模块通过 Verilog HDL 描述了计程车计费系统的相关功能。模块中定义了多个输入和输出端口，以便与其他模块进行数据交换。

- 输入端口:

1. clk：时钟输入，用于模块内部同步。

2. rst：复位信号，用于初始化和复位模块。

3. motor：表示计程车是否启动的信号。

4. op：操作模式，用于区分计程车的运营状态。

- 输出端口:

1. mil：表示里程数的输出。

2. dis：表示距离的输出。

3. timee：表示时间的输出。

4. sec：表示秒数的输出。

5. cost：表示费用的输出。

6. seg：表示费用的输出的分段。

7. codeout：表示输出的代码。

8. cnt：表示计数器输出。

- 模块实例化:

1. taximeter1：实例化了一个名为i1的模块，用于处理计程车的里程和距离计算，并与输入输出端口进行连接。

2. taximeter2：实例化了一个名为i2的模块，用于处理计程车的时间计算，并与输入输出端口进行连接。

3. taximeter3：实例化了一个名为i3的模块，用于计算费用，并与输入输出端口进行连接。

4. taximeter4：实例化了一个名为i4的模块，用于处理所有输入输出信息，并计算最终的费用，并与输入输出端口进行连接。

该模块通过多个子模块实例化，分别处理计程车的里程、时间以及费用计算，最终整合输出最终的费用信息。

taximeter0.v：

// module taximeter0(

// clk,clk1000

// );

// input clk;

// output reg clk1000;

// reg [15:0] cnt;

// initial

// begin

// cnt = 0;

// clk1000 = clk;

// end

// always@(posedge clk)

// begin

// if(cnt < 16'd25000-1)

// cnt <= cnt+1;

// else

// begin

// cnt <= 0;

// clk1000 <= ~clk1000;

// end

// end

// endmodule

module taximeter0 (

clk,clk1000

);

input clk;

output clk1000;

assign clk1000 = clk;

endmodule

设计原理：

这段 Verilog 代码定义了一个名为 taximeter0 的模块，其设计用途是用于将输入的时钟信号 clk 重新分配并输出为 clk1000 信号。

- 输入端口:

1. clk：时钟输入端口，作为输入的时钟信号。

- 输出端口:

1. clk1000：时钟输出端口，用于输出重新分配后的时钟信号。

- 模块功能描述:

1. 该模块内部使用 assign 语句，将输入端口 clk 直接赋值给输出端口 clk1000，因此，clk1000 的信号与 clk 的信号完全一致。这个 taximeter0 模块的设计比较简单，只是简单地将输入端口的时钟信号直接输出为输出端口的时钟信号。

taximeter1.v：

module taximeter1 (

clk,rst,motor,mil,dis

);

input clk;

input rst;

input motor;

output [7:0] mil;

output reg [17:0] dis;

parameter per = 2;

wire pedge;

wire nedge;

wire change;

reg [1:0] d;

initial

begin

dis = 0;

d = 2'b00;

end

always@(posedge clk)

begin

d <= {d[0],motor};

end

assign pedge = ~d[1] & d[0];

assign nedge = d[1] & ~d[0];

assign change = pedge | nedge;

always@(posedge clk, negedge rst)

begin

if(!rst)

dis <= 0;

else

if(change)

if(dis <= 18'd99\_997)

dis <= dis + per;

end

assign mil = dis/1000;

endmodule

设计原理：

这段 Verilog 代码定义了一个名为 taximeter1 的模块，其设计用途是处理计程车的里程和距离计算。

- 输入端口:

1. clk：时钟输入端口，作为输入的时钟信号。

2. rst：复位信号，用于初始化和复位模块。

3. motor：计程车启动信号。

- 输出端口:

1. mil：表示里程数的 8 位输出。

2. dis：表示距离的 18 位输出 (在代码中使用 reg 定义)。

- 模块参数:

1. per：初始化为常数 2，用于表示每计算一次距离增加的值。

- 内部信号:

1. pedge、nedge、change：分别表示上升沿、下降沿和信号变化的信号。

2. d：一个 2 位寄存器，用于保存上一时刻和当前时刻的 motor 信号。

- 初始块:

1. 使用 initial 块对 dis 和 d 进行了初始化。

- 时钟触发块:

1. 该模块有一个 always@(posedge clk) 块，在每个时钟上升沿触发。

2. 在时钟触发块内部，d 被更新为 {d[0], motor}，即将当前的 motor 信号存入寄存器 d 的低位，而将前一时刻的 d 寄存器高位保留。

- 信号计算块:

1. 使用 assign 语句计算 pedge、nedge 和 change 信号。

2. pedge 为上升沿信号 (~d[1] & d[0])。

3. nedge 为下降沿信号 (d[1] & ~d[0])。

4. change 为信号变化信号 (pedge | nedge)。

- 距离计算块:

1. 该模块有一个 always@(posedge clk, negedge rst) 块，在每个时钟上升沿以及复位信号负边沿触发。

2. 在块内部，使用条件语句判断是否要对 dis 进行更新。

3. 如果复位信号 rst 为低电平时，将 dis 初始化为 0。

4. 如果 change 为真 (即 motor 信号发生变化)，且 dis 小于等于 18'd99\_997，将 dis 增加 per 的值。

- 输出:

1. mil 通过将 dis 除以 1000 得到，表示计程车的里程数。

综上所述，该 taximeter1 模块根据输入的时钟信号、复位信号和计程车启动信号，以及内部计算逻辑，实现了计算计程车距离并输出里程数的功能。

taximeter2.v：

module taximeter2(

clk,rst,motor,timee,sec

);

input clk;

input rst;

input motor;

output reg [6:0] timee;

output reg [6:0] sec;

reg [16:0] cnt;

reg [10:0] cnt2;

wire pegde;

wire nedge;

wire change;

reg [1:0] d;

wire flag;

parameter max\_time = 17'd60000; // 1min (1KHz)

initial

begin

sec = 0;

cnt = 0;

cnt2 = 0;

d = 2'b00;

timee = 0;

end

always@(posedge clk)

begin

d <= {d[0],motor};

end

assign pegde = ~d[1] & d[0];

assign nedge = d[1] & ~d[0];

assign change = pegde | nedge;

always@(posedge clk)

begin

if(change || !rst)

cnt <= 0;

else if(cnt <= max\_time)

cnt <= cnt+1;

end

assign flag = (cnt <= max\_time);

always @(posedge clk)

begin

if(!rst)

begin

cnt2 <= 0;

sec <= 0;

timee <= 0;

end

else

begin

if(flag)

begin

if(cnt2 < 11'd100 - 1) // 1s

cnt2 <= cnt2+1;

else

begin

cnt2 <= 0;

if(sec < 7'd59)

sec <= sec+1;

else

begin

sec <= 0;

if(timee < 7'd59)

timee <= timee+1;

end

end

end

end

end

endmodule

设计原理：

这段 Verilog 代码定义了一个名为 taximeter2 的模块，其设计用途是处理计程车的时间计算。

- 输入端口:

1. clk：时钟输入端口，作为输入的时钟信号。

2. rst：复位信号，用于初始化和复位模块。

3. motor：计程车启动信号。

- 输出端口:

1. timee：表示分钟的 7 位输出。

2. sec：表示秒数的 7 位输出。

- 内部信号:

1. pegde、nedge、change：分别表示上升沿、下降沿和信号变化的信号。

2. d：一个 2 位寄存器，用于保存上一时刻和当前时刻的 motor 信号。

3. flag：一个用于表示计时是否达到最大时间的信号。

4. cnt：一个 17 位寄存器，用于计数。

5. cnt2：一个 11 位寄存器，用于计数。

- 参数:

1. max\_time：初始化为常数 17，表示最大计时时间。

- 初始块:

1. 使用 initial 块对 sec、cnt、cnt2、d 和 timee 进行了初始化。

- 时钟触发块 1:

1. 该模块有一个 always@(posedge clk) 块，在每个时钟上升沿触发。

2. 在时钟触发块 1 内部，d 被更新为 {d[0], motor}，即将当前的 motor 信号存入寄存器 d 的低位，而将前一时刻的 d 寄存器高位保留。

- 信号计算块:

1. 使用 assign 语句计算 pegde、nedge 和 change 信号。

2. pegde 为上升沿信号 (~d[1] & d[0])。

3. nedge 为下降沿信号 (d[1] & ~d[0])。

4. change 为信号变化信号 (pegde | nedge)。

- 时间计算块:

1. 该模块有一个 always@(posedge clk) 块，在每个时钟上升沿触发。

2. 在时间计算块内部，通过条件语句判断是否要对 cnt 进行更新。

3. 如果复位信号 rst 为低电平时，或者 change 信号为真，将 cnt 初始化为 0。

4. 否则，如果 cnt 小于等于 max\_time，将 cnt 增加 1。

- 标志计算:

1. 使用 assign 语句计算 flag 信号，判断 cnt 是否小于等于 max\_time。

- 时间增加块:

1. 该模块有一个 always@(posedge clk) 块，在每个时钟上升沿触发。

2. 在时间增加块内部，根据时钟信号和复位信号的状态，分别对 cnt2、sec 和 timee 进行递增。

3. 当 flag 为真时，如果 cnt2 小于 11'd100 - 1，将其增加 1。

4. 如果 cnt2 达到 11'd100 - 1，将其重置为 0，并判断 sec 是否小于 7'd59，若是，则 sec 增加 1。

5. 如果 sec 达到 7'd59，将其重置为 0，并判断 timee 是否小于 7'd59，若是，则 timee 增加 1。

综上所述，该 taximeter2 模块根据输入的时钟信号、复位信号和计程车启动信号，以及内部计算逻辑，实现了计算计程车时间并输出分钟和秒数的功能。

taximeter3.v：

module taximeter3(

mil,timee,cost

);

input [7:0] mil;

input [6:0] timee;

output reg [10:0] cost;

initial

cost = 11'd8;

always @(mil,timee)

begin

if(mil > 8'd3 && timee > 7'd3)

cost <= 2\*mil + timee - 1;

else if(timee > 7'd3)

cost <= 5 + timee;

else if(mil > 8'd3)

cost <= 2\*mil + 2;

else

cost <= 11'd8;

end

endmodule

设计原理：

这段 Verilog 代码定义了一个名为 taximeter3 的模块，其设计用途是根据里程和时间计算计程车的费用。

- 输入端口:

1. mil：8 位输入，表示计程车的里程数。

2. timee：7 位输入，表示计程车的时间数。

- 输出端口:

1. cost：11 位输出，表示计程车的费用。

- 初始块:

1. 使用 initial 块对 cost 进行了初始化，设置初始费用为 11'd8。

- 行为块:

1. 该模块有一个 always @(mil, timee) 块，在每次 mil 或 timee 变化时触发计算。

2. 根据不同的条件，通过条件语句判断计算出 cost 的值。

- 条件判断:

1. 如果 mil 大于 8'd3 并且 timee 大于 7'd3，cost 的计算公式为：cost = 2\*mil + timee - 1。

2. 如果只有 timee 大于 7'd3，cost 的计算公式为：cost = 5 + timee。

3. 如果只有 mil 大于 8'd3，cost 的计算公式为：cost = 2\*mil + 2。

4. 如果以上条件都不满足，即 mil 和 timee 都小于等于 3，cost 被重置为初始值 11'd8。

综上所述，该 taximeter3 模块根据输入的里程数和时间数，按照一定的条件判断规则，计算出计程车的费用并输出。

taximeter4.v：

module taximeter4(

clk,op,mil,timee,cost,seg,codeout,cnt

);

input clk;

input [1:0] op;

input [7:0] mil; // 0~99

input [6:0] timee; // 0~59

input [10:0] cost; // 0~999.0

output reg [7:0] seg;

output reg [7:0] codeout;

output reg [2:0] cnt;

wire [7:0] copy\_seg[2:0];

wire [7:0] copy\_codeout[2:0];

initial

begin

cnt = 0;

seg = 8'b0000\_0000;

codeout = 8'b0000\_0000;

end

always @(posedge clk)

begin

if(cnt < 3'd3)

cnt <= cnt+1;

else

cnt <= 0;

end

taximeter4\_0 i4\_0(clk,cnt,mil, copy\_seg[0], copy\_codeout[0]);

taximeter4\_1 i4\_1(clk,cnt,timee,copy\_seg[1], copy\_codeout[1]);

taximeter4\_2 i4\_2(clk,cnt,cost, copy\_seg[2], copy\_codeout[2]);

always @(posedge clk)

begin

case (op)

2'b00:

begin

seg <= copy\_seg[0];

codeout <= copy\_codeout[0];

end

2'b01:

begin

seg <= copy\_seg[1];

codeout <= copy\_codeout[1];

end

2'b10:

begin

seg <= copy\_seg[2];

codeout <= copy\_codeout[2];

end

default:

begin

seg <= seg;

codeout <= codeout;

end

endcase

end

endmodule

// mil 4λ 0~99

module taximeter4\_0 (

clk,cnt,mil,seg,codeout

);

input [2:0] cnt;

input clk;

input [7:0] mil;

output reg [7:0] seg;

output reg [7:0] codeout;

wire [3:0] dig [3:0];

wire [7:0] copy\_codeout [3:0];

assign dig[0] = mil%10;

assign dig[1] = mil/10;

assign dig[2] = 0;

assign dig[3] = 0;

taximeter5 i4\_0\_1(dig[0],copy\_codeout[0]);

taximeter5 i4\_0\_2(dig[1],copy\_codeout[1]);

taximeter5 i4\_0\_3(dig[2],copy\_codeout[2]);

taximeter5 i4\_0\_4(dig[3],copy\_codeout[3]);

always @(clk)

begin

case (cnt)

3'd0:

begin

seg <= 8'b0000\_0001;

codeout <= copy\_codeout[0];

end

3'd1:

begin

seg <= 8'b0000\_0010;

codeout <= copy\_codeout[1];

end

3'd2:

begin

seg <= 8'b0000\_0100;

codeout <= copy\_codeout[2];

end

3'd3:

begin

seg <= 8'b0000\_1000;

codeout <= copy\_codeout[3];

end

default:

begin

seg <= 8'bx;

codeout <= 8'bx;

end

endcase

end

endmodule

// timee 2λ

module taximeter4\_1 (

clk,cnt,timee,seg,codeout

);

input [2:0] cnt;

input clk;

input [6:0] timee;

output reg [7:0] seg;

output reg [7:0] codeout;

wire [3:0] dig [1:0];

wire [7:0] copy\_codeout[1:0];

assign dig[0] = timee%10;

assign dig[1] = timee/10;

taximeter5 i4\_1\_1(dig[0],copy\_codeout[0]);

taximeter5 i4\_1\_2(dig[1],copy\_codeout[1]);

always@(clk)

begin

if(cnt == 3'd0 || cnt == 3'd2)

begin

seg <= 8'b0000\_0001;

codeout <= copy\_codeout[0];

end

else if(cnt == 3'd1 || cnt == 3'd3)

begin

seg <= 8'b0000\_0010;

codeout <= copy\_codeout[1];

end

end

endmodule

// cost 4λ

module taximeter4\_2 (

clk,cnt,cost,seg,codeout

);

input [2:0] cnt;

input clk;

input [10:0] cost;

output reg [7:0] seg;

output reg [7:0] codeout;

wire [3:0] dig [3:0];

wire [7:0] copy\_codeout[3:0];

assign dig[0] = 0;

assign dig[1] = cost%10;

assign dig[2] = (cost%100)/10; // ʮλ

assign dig[3] = cost/100;

taximeter5 i4\_2\_1(dig[0],copy\_codeout[0]);

taximeter5 i4\_2\_2(dig[1],copy\_codeout[1]);

taximeter5 i4\_2\_3(dig[2],copy\_codeout[2]);

taximeter5 i4\_2\_4(dig[3],copy\_codeout[3]);

always @(clk)

begin

case (cnt)

3'd0:

begin

seg <= 8'b0000\_0001;

codeout <= copy\_codeout[0];

end

3'd1:

begin

seg <= 8'b0000\_0010;

codeout <= copy\_codeout[1];

codeout[0] <= 1;

end

3'd2: // ʮλ

begin

seg <= 8'b0000\_0100;

codeout <= copy\_codeout[2];

end

3'd3:

begin

seg <= 8'b0000\_1000;

codeout <= copy\_codeout[3];

end

default:

begin

seg <= 8'bx;

codeout <= 8'bx;

end

endcase

end

endmodule

设计原理：

这段 Verilog 代码定义了一个名为 taximeter4 的模块，其中还包含了三个子模块 taximeter4\_0、taximeter4\_1 和 taximeter4\_2。这可以被视为一个包含多个模块的顶层设计，可能是为了实现更复杂的功能而将功能拆分到不同模块中实现。

该 taximeter4 模块包含了一些输入输出端口，其中 clk 是时钟输入，op、mil、timee 和 cost 分别表示操作符、里程、时间和费用，seg 和 codeout 是用于输出七段数码管显示的信号，cnt 表示计数器。

顶层模块的功能主要包含以下部分：

1. cnt、seg 和 codeout 的初始赋值；

2. 使用时钟边沿触发的计数器 cnt；

3. 三个子模块实例化 i4\_0、i4\_1 和 i4\_2，分别对应 taximeter4\_0、taximeter4\_1 和 taximeter4\_2；

4. 根据操作符 op 的不同，选择相应子模块的输出作为 seg 和 codeout 的输出。

每个子模块（taximeter4\_0、taximeter4\_1 和 taximeter4\_2）都会根据输入的数据进行一些处理，然后将结果输出给 seg 和 codeout。

这种设计方式将计算和逻辑处理分解到不同模块中，并通过顶层模块实现对这些子模块的组合和控制，从而实现了更加模块化的设计思路。

需要注意的是，在 Verilog 代码中使用到的 always @(posedge clk) 表示使用时钟的上升沿触发，而 always @(clk) 则表示该代码块会响应 clk 的任何变化。

taximeter5.v：

module taximeter5(

indec,codeout

);

input[3:0] indec;

output reg [7:0] codeout;

always @(indec) begin

case(indec)

4'd0: codeout = 8'b1111\_1100;

4'd1: codeout = 8'b0110\_0000;

4'd2: codeout = 8'b1101\_1010;

4'd3: codeout = 8'b1111\_0010;

4'd4: codeout = 8'b0110\_0110;

4'd5: codeout = 8'b1011\_0110;

4'd6: codeout = 8'b1011\_1110;

4'd7: codeout = 8'b1110\_0000;

4'd8: codeout = 8'b1111\_1110;

4'd9: codeout = 8'b1111\_0110;

default: codeout = 8'bx;

endcase

end

endmodule

设计原理：

这段 Verilog 代码定义了一个名为 taximeter5 的模块，它有两个端口 indec 和 codeout，其中 indec 是输入端口，表示一个四位的二进制数，而 codeout 是输出端口，表示一个八位的二进制数。

在 always @(indec) 的代码块中，根据输入的 indec 的不同值，使用 case 语句对 codeout 进行赋值操作。这里的 case 语句会根据 indec 的值在不同情况下执行不同的代码。

具体来说，当 indec 的值为 4 位二进制数 0、1、2、3、4、5、6、7、8、9 中的某一个时，分别对应不同的情况，比如当 indec 的值为 4'b1，也就是二进制的 1 时，将 codeout 的值赋为 8'b0110\_0000。这样的设计原理可以通过一个比较简单的逻辑实现将不同的 indec 值转化为相应的 codeout 值。

而当 indec 的值不满足以上的情况时，也就是没有匹配到任何一个 case 条件时，会进入 default 分支，将 codeout 的值赋为 8'bx，表示不确定的或者无效的值。

通过这样的设计原理，可以实现根据输入的 indec 值，将对应的数字编码存储在 codeout 中，以便后续的处理或显示。

test\_taximeter.v：

timescale 1ms/1ms

module tset\_taximeter();

reg clk;

reg rst;

reg motor;

reg [1:0] op;

// wires

wire [7:0] codeout;

wire [10:0] cost;

wire [7:0] mil;

wire [17:0] dis;

wire [7:0] seg;

wire [6:0] timee;

wire [6:0] sec;

wire [2:0] cnt;

taximeter i1 (

.clk(clk),

.rst(rst),

.codeout(codeout),

.cost(cost),

.mil(mil),

.dis(dis),

.motor(motor),

.op(op),

.seg(seg),

.timee(timee),

.sec(sec),

.cnt(cnt)

);

reg [23:0] cnt1;

reg [26:0] cnt2;

// reg [8:0] cnt3;

// reg [6:0] sec;

reg [5:0] num;

initial

begin

clk = 0;

rst = 1;

motor = 0;

op = 0;

cnt1 = 0;

cnt2 = 0;

// cnt3 = 0;

// sec = 0;

num = 6'd24; // 60km/h

#120000 // 2min

num = 6'd20; // 72km/h

#3120000

rst = 0;

#1000000

rst = 1;

end

always # 5

begin

clk <= ~clk;

if(cnt1 < num-1) // 60km/h

cnt1 <= cnt1+1;

else

begin

cnt1 <= 0;

motor <= ~motor;

end

if(cnt2 < 27'd2000-1) // 10s

cnt2 <= cnt2+1;

else

begin

cnt2 <= 0;

if(op < 2)

op <= op+1;

else

op <= 0;

end

// if(cnt3 < 9'd200 - 1)

// cnt3 <= cnt3+1;

// else

// begin

// cnt3 <= 0;

// if(sec < 7'd59)

// sec <= sec+1;

// else

// sec <= 0;

// end

end

endmodule

设计原理：

这段 Verilog 代码定义了一个名为 tset\_taximeter 的模块以及其中的一些寄存器、线网和信号连接以及一个名为 taximeter 的实例 i1。下面是代码的详细解释：

1. timescale 1ms/1ms 定义了时间单位，这里表示时钟周期为 1 毫秒。

2. 模块内定义了一些寄存器（reg）和线网（wire），分别用于时钟信号 clk、复位信号 rst、电机信号 motor，操作符 op 等。其中使用了不同位数的寄存器来存储不同长度的信号。

3. 实例化了一个名为 i1 的 taximeter 模块，将定义好的端口连接到该实例的对应端口上。

4. 定义了一些局部的寄存器，如 cnt1、cnt2 和 num 用于实现一些逻辑控制。

5. 在 initial 代码块中，对部分寄存器进行了初始化，设置了一些初始值，并使用 # 符号定义了时间延迟。具体来说：首先，设置了一系列初始值，然后等待了 120000 个时钟周期（约 120 毫秒），接着修改了 num 的值，再等待了 3120000 个时钟周期（约 3120 毫秒），最后修改了 rst 的值，并再等待了 1000000 个时钟周期（约 1000 毫秒）。这样的初始化过程用于模拟系统上电后的初始化过程。

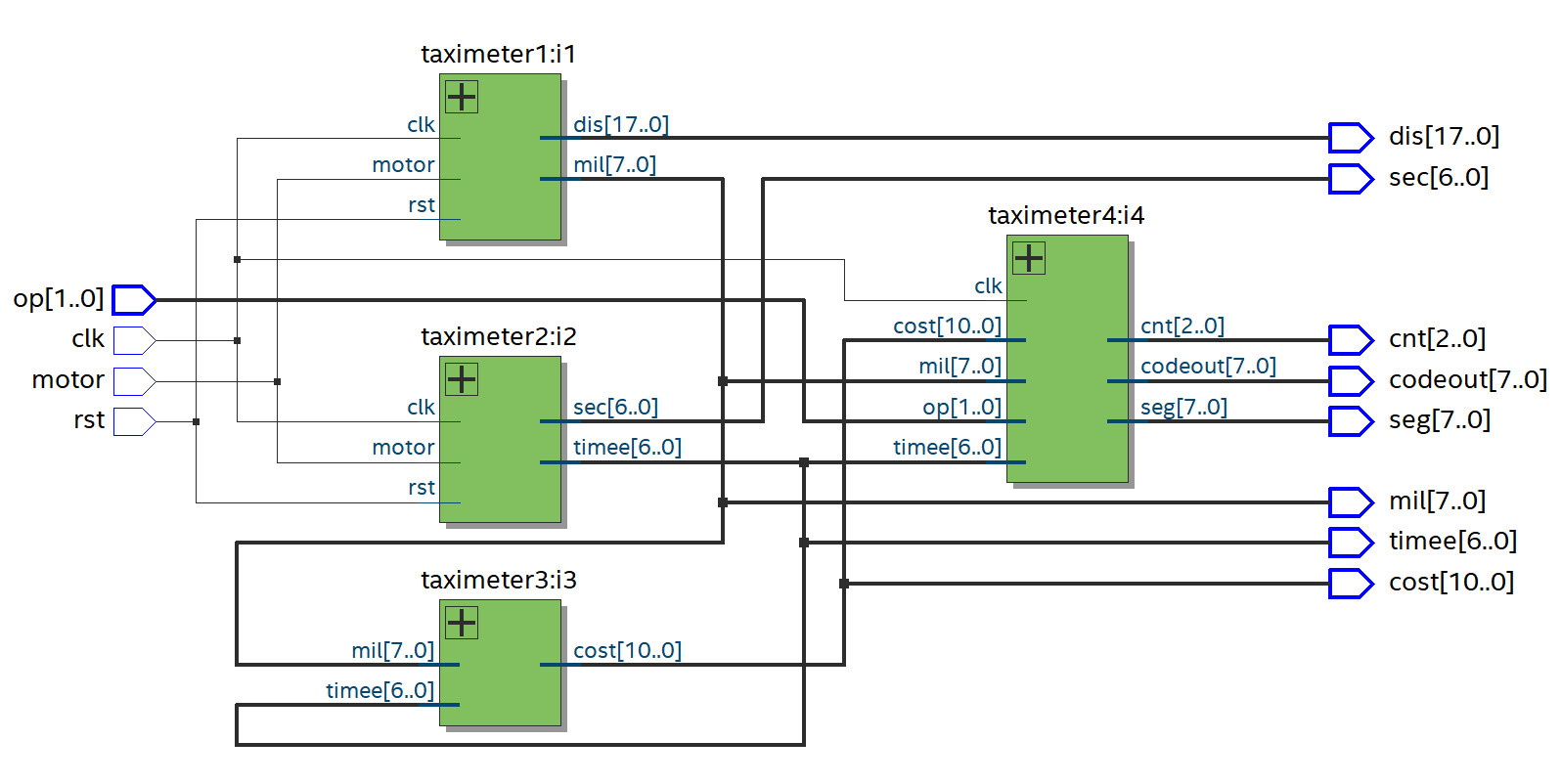
6. 在 always #5 的代码块中，实现了一个基于时钟的逻辑控制过程。在每 5 个时钟周期中，时钟信号 clk 取反，同时对 cnt1 和 cnt2 进行递增判断和控制，以及对 motor 和 op 状态的更新。

总体来说，这段代码实现了一个简单的模拟系统时钟控制逻辑，通过对寄存器的初始化和对不同信号的控制，模拟了系统上电初始化和时钟控制的过程。

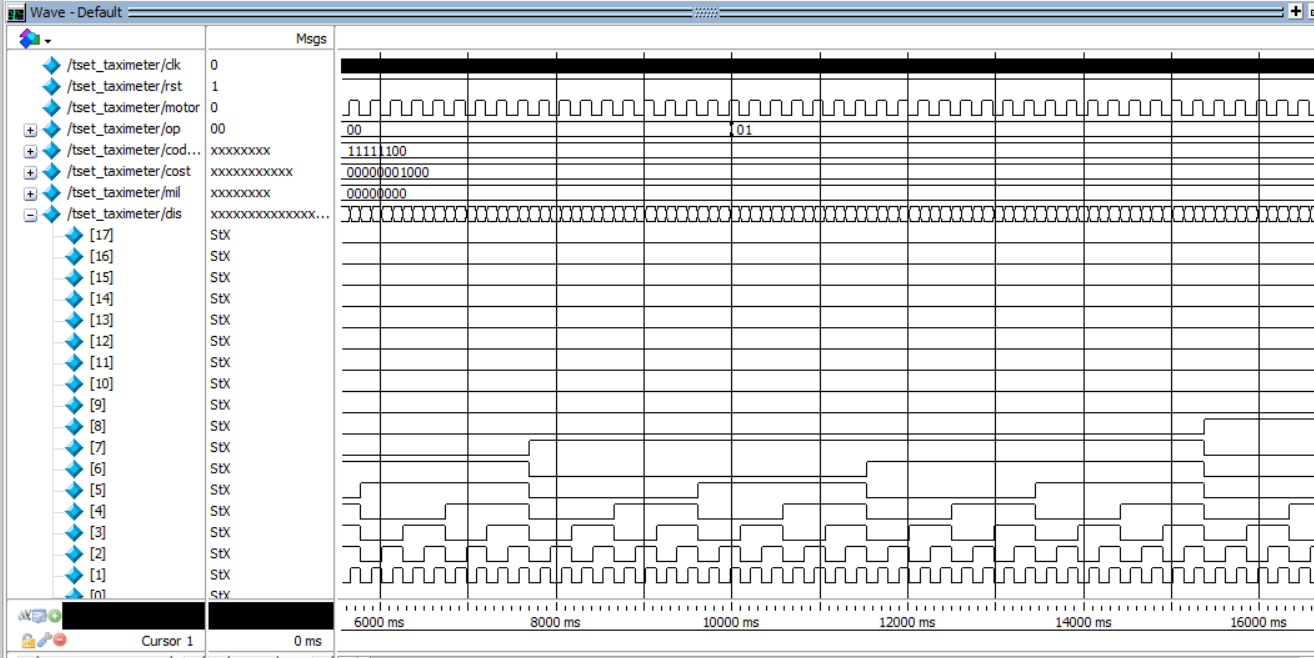
引脚分配：

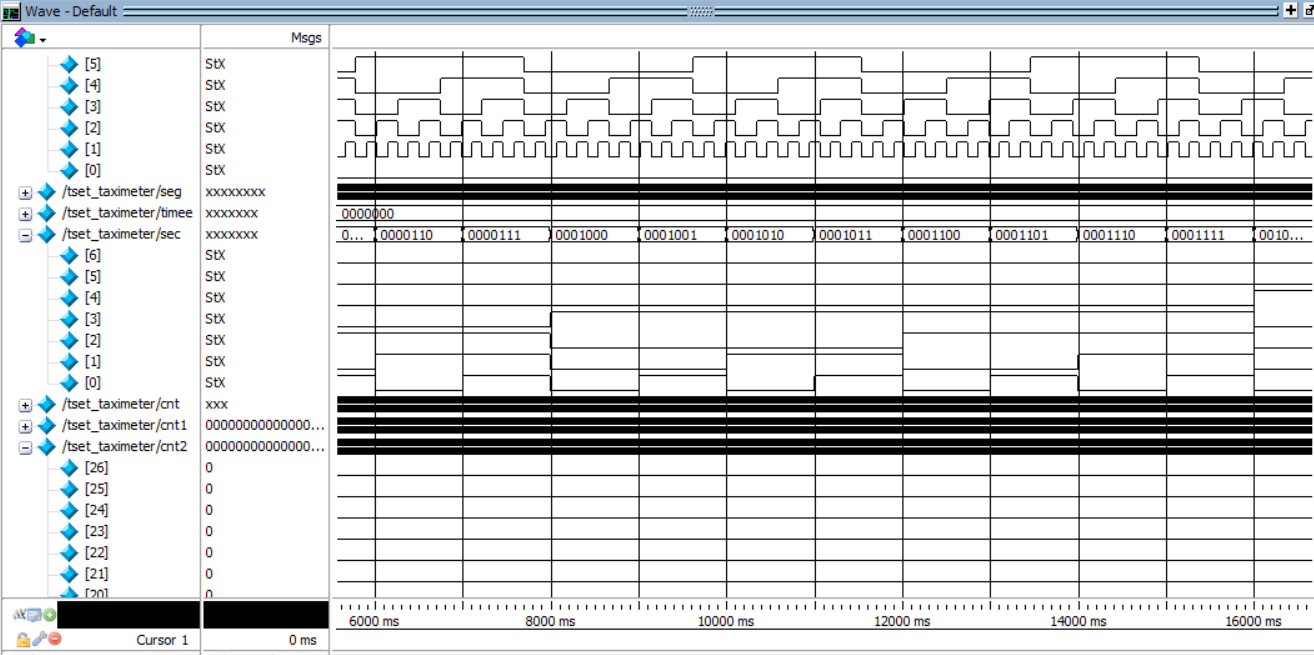
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **信号名** | **主板器件** | **PIN** |  | **信号名** | **主板器件** | **PIN** |
| **clk** | **CLK0** | **PIN\_88** | **codeout[7]** | **a** | **PIN\_112** |
| **rst** | **SW0** | **PIN\_24** | **codeout[6]** | **b** | **PIN\_100** |
| **motor** | **ECLK** | **PIN\_23** | **codeout[5]** | **c** | **PIN\_104** |
| **op[1]** | **SW2** | **PIN\_30** | **codeout[4]** | **d** | **PIN\_111** |
| **op[0]** | **SW1** | **PIN\_31** | **codeout[3]** | **e** | **PIN\_106** |
| **seg[7]** | **SEG7** | **PIN\_120** | **codeout[2]** | **f** | **PIN\_110** |
| **seg[6]** | **SEG6** | **PIN\_113** | **codeout[1]** | **g** | **PIN\_103** |
| **seg[5]** | **SEG5** | **PIN\_121** | **codeout[0]** | **h** | **PIN\_105** |
| **seg[4]** | **SEG4** | **PIN\_114** | **seg[0]** | **SEG0** | **PIN\_119** |
| **seg[3]** | **SEG3** | **PIN\_125** |  |  |  |
| **seg[2]** | **SEG2** | **PIN\_115** |  |  |  |
| **seg[1]** | **SEG1** | **PIN\_126** |  |  |  |

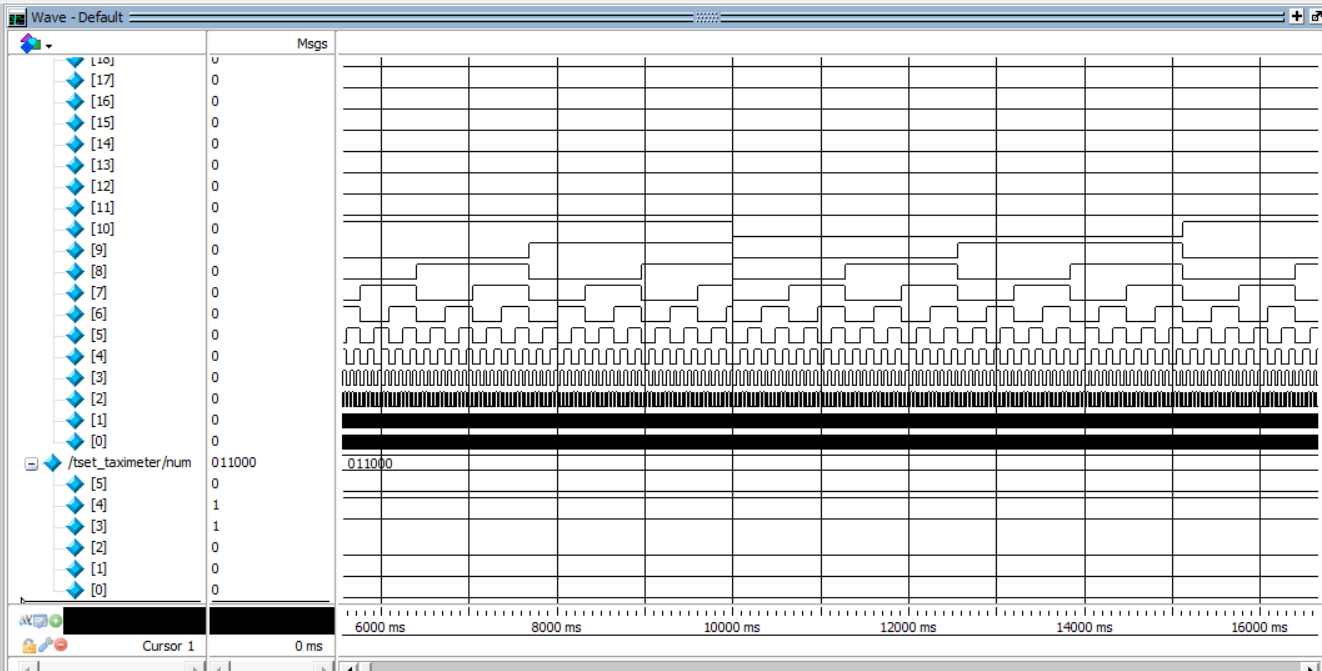
RTL图：

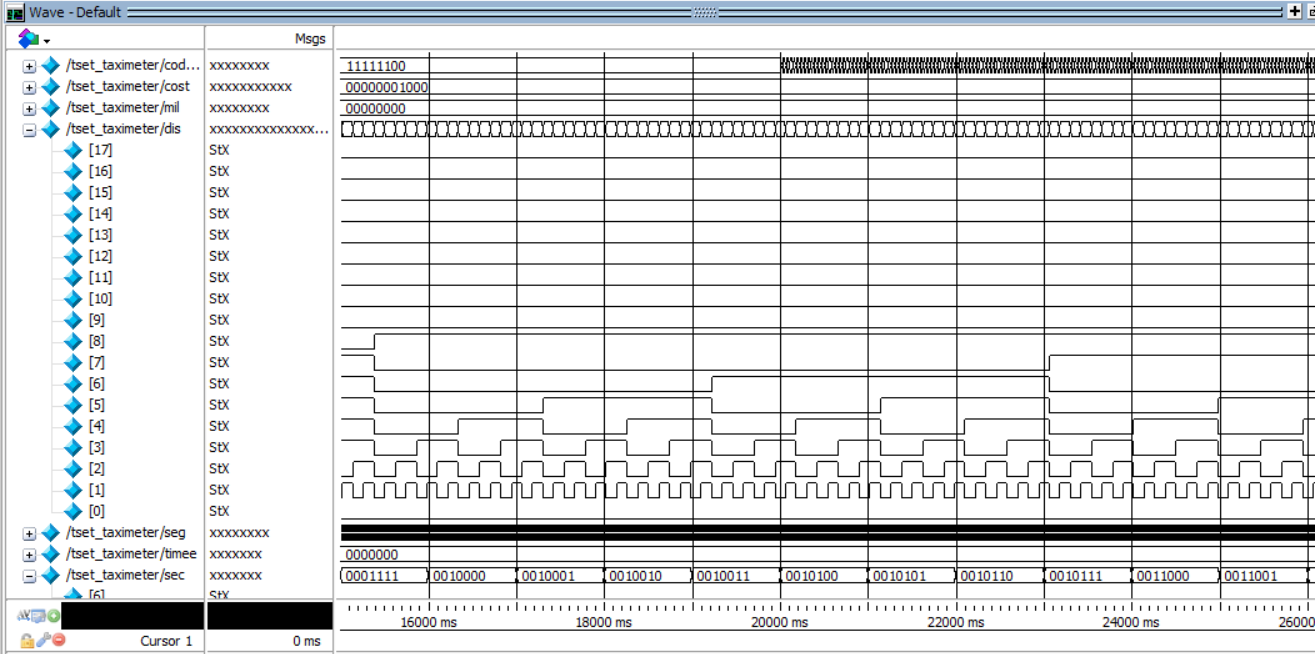


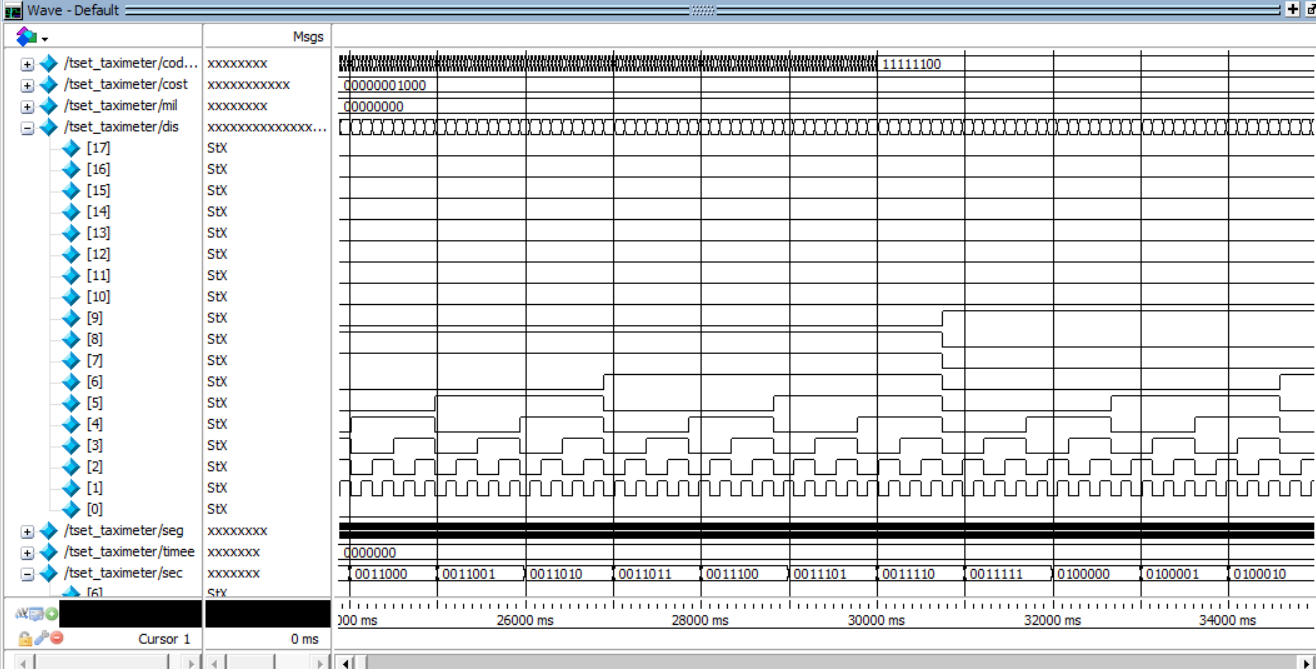
仿真波形：



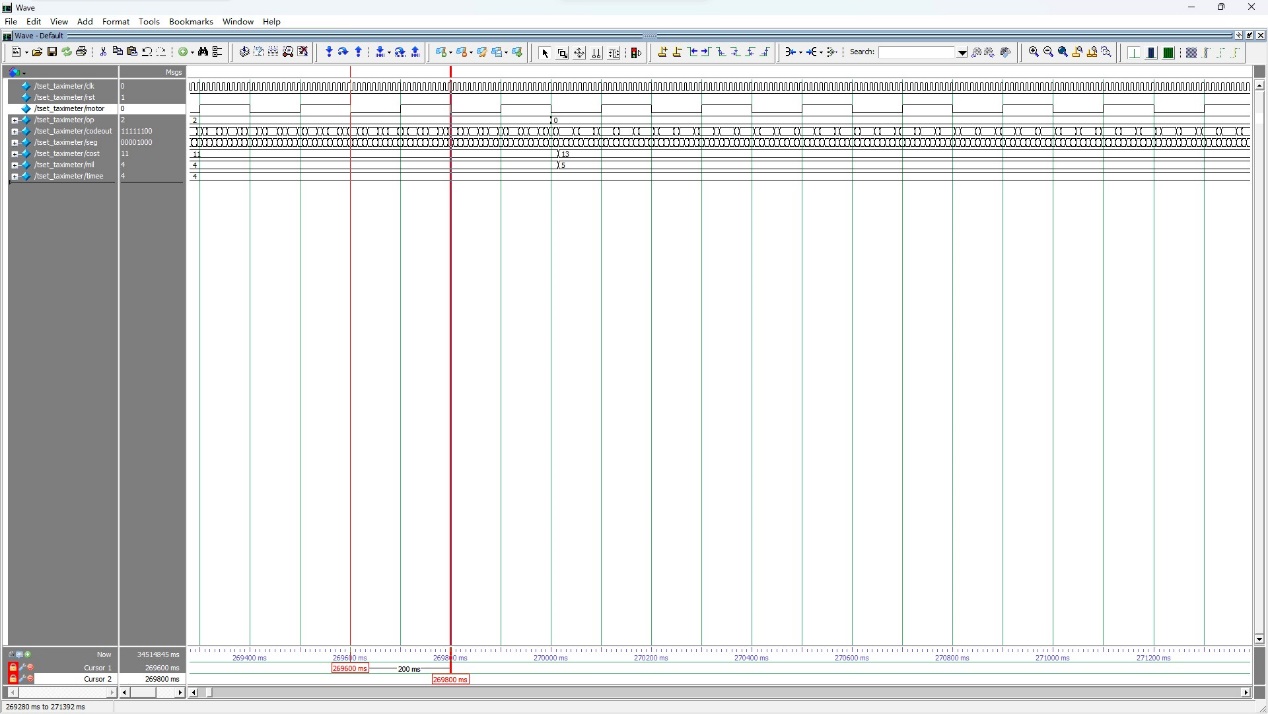


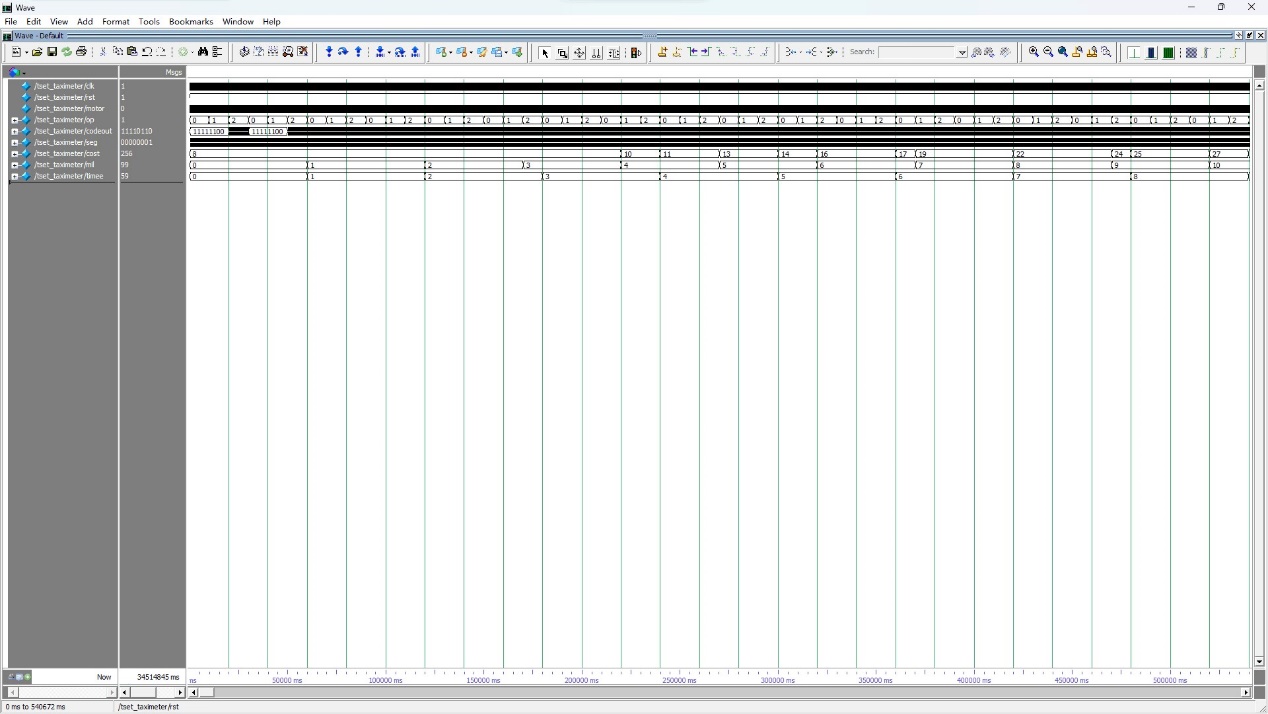






3. 总体仿真与分析





1. **心得体会**

我们两位成员一起完成了Quartus出租车计价器的功能制作，每个模块的代码都是我们共同努力的结果。通过这个项目，我们两都获得了许多宝贵的经验和体会。

例如，taximeter5.v模块是整个出租车计价器的核心模块。在这个模块中，我们使用case语句根据输入的indec值，将对应的数字编码存储在codeout中。这个模块的设计原理非常简单，但需要精确地处理输入和输出之间的转换。通过我们两的合作和讨论，我们确保了模块的功能正确性和可靠性。

再如，tset\_taximeter.v模块是整个系统的控制模块。在这个模块中，我们定义了时钟信号clk、复位信号rst、电机信号motor、操作信号op等，并使用了一系列的寄存器和计数器来实现时钟的控制和状态的更新。通过这个模块，我们能够模拟出计价器的工作过程，并确保各个功能模块按照正确的时序进行相应的操作。在编写这个模块的过程中，我们共同讨论，制定了一套合理的时钟控制策略，以确保整个计价器的稳定性和可靠性。

在项目的过程中，我们两个成员积极合作，相互学习和支持，共同解决了许多技术和设计上的问题。我们共同分工，相互协作，确保了项目的顺利进行。通过相互合作，我们都提高了自己的编程技能和问题解决能力，还学习到了团队合作和沟通的重要性。我们共同面对挑战，克服了各种困难，最终完成了这个功能制作。

这次项目的完成对我们两来说都是一次重要的成长机会。通过参与这个项目，我们深入了解了数字电路设计和Verilog编程的原理和应用。学会了如何合理划分模块功能，并使用Verilog语言实现相应的代码。提高了对FPGA开发工具的使用熟练度，并学会了如何进行调试和优化。最重要的是，学会了如何与团队成员合作，理解彼此的需求，并共同努力完成一个共同的目标。

综上所述，完成Quartus出租车计价器的功能制作是团队成员（虽然只有两个人）共同努力的结果。通过这个项目，我们不仅拓宽了自己的技术视野，同时提升了团队合作和沟通能力。这次经历对我们而言都是一次宝贵的学习机会，我们都将继续努力提升自己，在未来的项目中发挥更大的作用。

1. **参考文献**

1、http://t.csdnimg.cn/3m7Xu

2、http://t.csdnimg.cn/b1oEg

1. **附件Quartus工程文件夹**

见taximeter文件夹