Lab 3 Report - Design of the Taxi Meter

Lab Targets

This lab will design a taxi meter. The detailed functions are following.

The starting price is 13 yuan, and the first three kilometres are the starting price range. The distance cost is not calculated separately, but the low-speed driving fee is still charged.

When the distance is bigger than 3km, the money adds 2.3 yuan every kilometre.

When the money is bigger than 50 yuan, the money of every kilometre becomes 3.3 yuan per kilometre.

If the taxi in low speed status, every 5 minutes will adds 2.3 yuan to total cost.

There are three states. ‘start’, ‘stop’ and ‘pause’. ‘start’ means money sets to initial state and distance increases from zero. ‘stop’ means that money and distance is becomes zero. When the car pauses, do not count money and distance for this time.

The distance is showing on 5 7-segment LEDs, which has two decimal places and money showing on 4 7-segment LEDs which has one decimal place.

Circuit Diagram

Code and Comments

Top Model - Lab3.v

Control Module - taxi_control.v

In this module, we used the wheel speed(rad/s) to determine the speed of the taxi. We assume a sensor on the wheel, every time the wheel turns a cycle, we get a pulse in **wheel_clk**. All we need to do is count the number of turns the wheel has made in a given amount of time, and then divide the number of turns by that time, which is the speed of the wheel.

```
module taxi_control(clk, wheel_clk, stop, start, pause, low_speed, high_speed, pause_state, stop_state);
    input clk, wheel_clk, stop, start, pause;
    output reg low_speed, high_speed, pause_state, stop_state;

    reg [4:0]clk_counter, wheel_counter;
    reg control;//判断是否经过固定时间
    reg [4:0]judge;

    initial begin
        clk_counter = 0;
        wheel_counter = 0;
        judge = 0;
        control = 0;
    end
```

The following are the meanings and uses of some variables:

1. clk_counter: Recording the number of posedge in **clk** (how many clock cycles have been went through).
2. wheel_counter: Recording the number of posedge in **wheel_clk** (how many times did the wheel turn).
3. control: Determine whether the number of goals of clock cycles(10, which I will introduce later) have passed or not. If true, set **control** to HIGH, else set **control** to LOW.
4. judge: Wheel speed.

```
    always@(posedge clk)begin
        if(control) control = 0;
        clk_counter = clk_counter + 1;
        if(clk_counter == 10)begin
            control = 1;
            clk_counter = 0;
        end
    end


    always@(posedge wheel_clk)begin
        if(!control) wheel_counter = wheel_counter +1 ;
        else begin
            judge = wheel_counter;
            wheel_counter = 0;
        end
    end
```

Every time **clk_counter** goes up to 10, which is one second(as we set in this lab), we stop updating **wheel_counter** to read **wheel_counter**, which is equal to wheel speed(because the period is 1s). In this condition, judge is equal to **wheel_counter**.

Finally, reset **clk_counter**, **wheel_counter**, and **control**, in order to get the wheel speed of next second.

```
    always@(posedge clk)begin
        if(stop)begin
            low_speed = 0;
            high_speed = 0;
            pause_state = 0;
            stop_state = 1;
        end
        else if(start)begin
            stop_state = 0;
            if(pause)begin
                high_speed = 0;
                low_speed = 0;
                pause_state = 1;
            end
            else begin
                pause_state = 0;
                if(judge >= 10)begin
                    low_speed = 0;
                    high_speed = 1;
                end
                else begin
                    high_speed = 0;
                    low_speed = 1;
                end
            end
        end
    end
```

In this module, we set the speed at ten as the dividing line between low speed and high speed. If **judge** >= 10, we set high_speed to 1, else, we set low_speed to 1.

Distance Calculation Module - taxi_distance_ptime.v

```
module taxi_distance_ptime(high_speed, low_speed, pause_state, stop_state, wheel_clk, clk, distance, low_time);
input high_speed, low_speed, pause_state, stop_state, wheel_clk, clk;
output reg [31:0] low_time, distance;

reg [4:0] wheel_counter2;
reg [10:0] clk_counter2; //low speed time

initial begin
    low_time = 0;
    distance = 0;
    wheel_counter2 = 0;
    clk_counter2 = 0;
end
```

This module named `taxi_distance_ptime`, has six input ports and two output ports.

1. high_speed:
2. low_speed: This signal from control module, when taxi in low speed status this signal is HIGH, otherwise is LOW.
3. pause_state: When this signal is HIGH, taxi meter not calculate any money and distance.

4. stop_state: When this signal is HIGH, it means that the passenger has got off and the amount and mileage will be cleared.
5. wheel_clk: This signal will generate a high level every time the wheel makes one turn. We use this signal to calculate the speed of the vehicle.
6. clk: Clock signal.
7. low_time: 32-bit register type output, which is taxi low speed time.
8. distance: 32-bit register type output, which is distance of taxi.

There are two intermediate variables named **wheel_counter2** and **clk_counter2**. The first variable is used to calculate how many laps the wheel has traveled, and the second variable is used to measure the low speed of the taxi.

At the initial state, we will set all variables as 0.

```
always@(posedge wheel_clk or posedge stop_state)begin
   if(stop_state) begin
        distance = 0;
        wheel_counter2 = 0;
    end
    else begin
        if(!pause_state)begin
            wheel_counter2 = wheel_counter2 + 1;
                if(wheel_counter2 == 10)begin //10圈轮子7米
                    distance = distance + 7;
                    wheel_counter2 = 0;
                end
        end
    end

end
```

This part used for calculate distance. If the **stop_state** signal is HIGH, indicating that the taxi is not carrying passengers, we will clear the **distance** and **wheel_counter2**.

When the taxi is not in pause mode, **wheel_counter2** calculates the number of laps the wheel has turned. We specify that the wheel has turned 10 times and the car travels 7 meters. When **wheel_counter2** reaches 10, we add 7 to the **distance** and clear **wheel_counter2**.

```
always@(posedge clk or posedge stop_state)begin
    if(stop_state)begin
        low_time = 0;
        clk_counter2 = 0;
    end
    else begin
        if(low_speed) clk_counter2 = clk_counter2 + 1;
        low_time = clk_counter2/100; //100个cycle是1分钟
    end
end
endmodule
```

This part used for calculate low speed time. If the **stop_state** signal is HIGH, indicating that the taxi is not carrying passengers, we will clear the **low_time** and **clk_counter2**.

When **low_speed** signal is HIGH, which means taxi has low speed, **clk_counter2** will add 1 each clock cycle. We specify that 100 clock cycles is 1 minutes. So, low time is equal to **clk_counter2** divided by 100.

Price calculation Module - taxi_money.v

LED Module - LED2s.v

In order to display the distance that the taxi moved during the trip, as well as the fare that passengers need to pay, we assign to utilize LEDs. it is noticeable that what we have designed is dynamic scanning LEDs.

```
module LED2s(distance_out, money, display, scan, clk2, clk, dp);
    input [16:0] distance_out, money;
    input clk2, clk;//clk2 用来扫描 clk2要远远快于clk
    output reg [6:0] display;
    output reg [8:0] scan;
    output reg dp;
    reg [10:0] X1_distance, X2_distance, G_distance, S_distance,B_distance;
    reg [10:0] X1_money, G_money, S_money, B_money;
    reg [3:0] chos, data;

    parameter BLANK = 7'b0000000;
    parameter ZERO  = 7'b1111110;
    parameter ONE   = 7'b0110000;
    parameter TWO   = 7'b1101101;
    parameter THREE = 7'b1111001;
    parameter FOUR  = 7'b0110011;
    parameter FIVE  = 7'b1011011;
    parameter SIX   = 7'b1011111;
    parameter SEVEN = 7'b1110000;
    parameter EIGHT = 7'b1111111;
    parameter NINE  = 7'b1111011;

    initial begin
        chos = 10;
        scan = 'b000000000;
        display = BLANK;
    end

    always@(posedge clk)begin
        X2_distance = distance_out%100/10;
        X1_distance = distance_out%1000/100;
        G_distance =  distance_out%10000/1000; //distance 米，要变成千米
        S_distance =  distance_out%100000/10000;
        B_distance =  distance_out%1000000/100000;

        X1_money = money%10;
        G_money = money%100/10;
        S_money = money%1000/100;
        B_money = money%10000/1000;
    end
```

We need 9 LEDs to demonstrate different places of the value of distance and money, for example **X2_distance** represents the second place after the decimal point, **X1_distance** represents the first place after the decimal point, **G_distance**----ones place, **S_distance**'---tens place, **B_distance**---hundreds place, which have the same meaning of **X1_money**, **G_money**, **S_money**, **B_money**.

Note: all of these variables I mentioned above(**X2_distance X1_distance G_distance S_distance**.....**B_money**) called intermediate variables.

As you can see in the code, there are four inputs---**distance_out**, **money**, **clk**, **clk2**. **distance_out** and **money** are used to indicate the actual distance as well as the actual fare needed to display, **clk** is used to control refreshing data----as the posedge of clk occurs the value of each intermediate variable discussed before will be refreshed. **clk2** is used to control dynamic scanning of LEDs----as the posedge of clk2 occurs the module will choose the next LED to show corresponding number ,thus '**clk2**' is much faster than **clk**----at least it should satisfy that before the next posedge of clk (refreshing data), all of LEDs should display their value once. While, for the three outputs---**display**, **scan**, **dp**, "display" is used to output 7-segment decoding instructions corresponding with the number stored in each intermediate variable, "scan" is used to output the LED address utilizing 9-bit one-hot code form 000000001'b to 100000000'b and "dp" is used to display status of decimal point on the LED.

```
    always@(data)begin
        case(data)
            0:begin display = ZERO; end
            1:begin display = ONE;  end
            2:begin display = TWO;  end
            3:begin display = THREE; end
            4:begin display = FOUR; end
            5:begin display = FIVE; end
            6:begin display = SIX;  end
            7:begin display = SEVEN;end
            8:begin display = EIGHT;end
            9:begin display = NINE; end
            default:begin display = BLANK; end
        endcase
    end



    always@(posedge clk2)begin
        if(chos < 8) chos = chos + 1;
        else chos = 0;
    end

    always@(chos)begin
        case(chos)
            0:begin data = X2_distance;dp = 0;scan = 'b000000001; end
            1:begin data = X1_distance;dp = 0;scan = 'b000000010; end
            2:begin data = G_distance;dp = 1;scan = 'b000000100;  end
            3:begin data = S_distance;dp = 0;scan = 'b000001000;  end
            4:begin data = B_distance;dp = 0;scan = 'b000010000;  end
            5:begin data = X1_money;dp = 0;scan = 'b000100000;    end
            6:begin data = G_money;dp = 1;scan = 'b001000000;     end
            7:begin data = S_money;dp = 0;scan = 'b010000000;     end
            8:begin data = B_money;dp = 0;scan = 'b100000000;     end
            default:begin data = 10;dp = 0;scan = 'b000000000;     end
        endcase
    end
endmodule
```

Variable **chos** and **data** can help us to determine which LED will be turned on and which number will be shown on LED.

If we want LEDs to show the correct number, corresponding 7-segment code need to be transferred to them exactly

Testbench and Wave

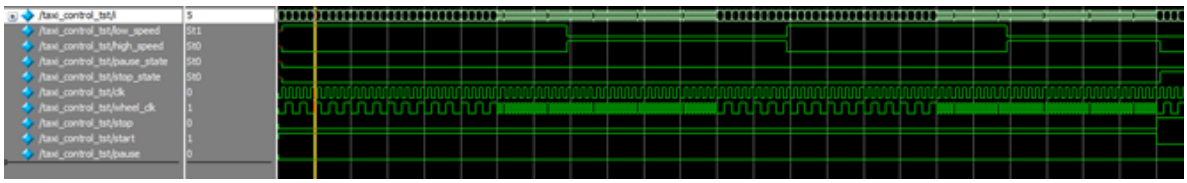
Control Module Testbench

```
`timescale 1 ps/ 1 ps
module taxi_control_vlg_tst();
// constants
// general purpose registers
reg eachvec;
// test vector input registers
reg clk;
reg pause;
reg start;
reg stop;
reg wheel_clk;
// wires
wire high_speed;
wire low_speed;
wire pause_state;
wire stop_state;

// assign statements (if any)
taxi_control i1 (
// port map - connection between master ports and signals/registers
    .clk(clk),
    .high_speed(high_speed),
    .low_speed(low_speed),
    .pause(pause),
    .pause_state(pause_state),
    .start(start),
    .stop(stop),
    .stop_state(stop_state),
    .wheel_clk(wheel_clk)
);
initial
begin
// code that executes only once
// insert code here --> begin
    clk = 0;
    wheel_clk = 0;
    stop = 0;
    start = 1;
    pause = 0;
    forever #50 clk = ~clk;
// --> end
$display("Running testbench");
end

initial
begin
    #4000 stop = 1;
            start = 0;
    #8000 stop = 0;
```

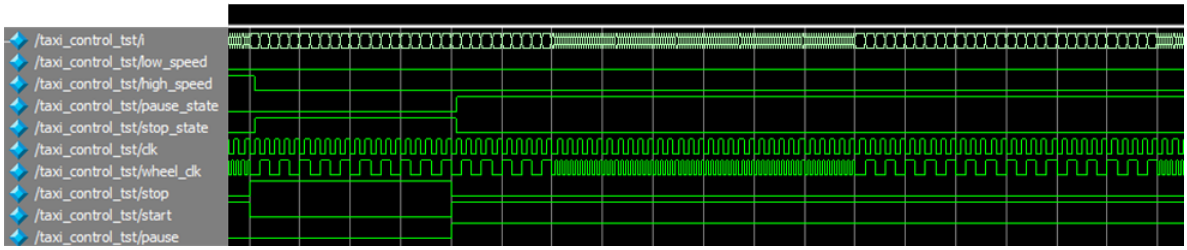
```
            start = 1;
            pause = 1;
end

always
// optional sensitivity list
// @(event1 or event2 or .... eventn)
begin
// code executes for every event on sensitivity list
// insert code here --> begin
    for(i=0;i<10;i=i+1)
        begin
            #100 wheel_clk = ~wheel_clk;
        end

    for(i=0;i<20;i=i+1)
        begin
            #50 wheel_clk = ~wheel_clk;
        end
@eachvec;
// --> end
end
endmodule
```

At first, we using two loop to generate wheel pules, the first loop is testing low speed function. The second loop is testing normal (high speed) function.

The initial value of low_speed is 1. In first loop, judge = 5 < 10, low_speed = 1. In second loop, judge = 20 > 10, so since 10 clock cycles after 3000ps(4000ps), high_speed turns to 1.



Then, we testing *stop*, *start*, and *pause* function, the wave as following:



All information changes are correctly displayed on the waveform.

LED Module Testbench

In the testbench, the main mission is to check if LEDs can display the value we want and if the dynamic scanning can work perfectly, the code of testbench as shown below:

```
`timescale 1ns/10ps
module LED2s_test();
reg [16:0] distance_out, money;
reg clk2,clk;
wire [8:0] scan;
```

```
wire dp;
wire [6:0]display;
LED2s L1 (distance_out, money, display, scan, clk2, clk, dp);

always begin
    #50 clk = ~ clk;
end

always begin
    #2 clk2 = ~ clk2;
end

initial begin
    clk = 0;
    clk2 = 0;
    distance_out = 12345;
    money = 6666;
    #200 distance_out = 78965;
    money = 5555;
    #350 $finish;
end

initial $monitor
($time,
"current distance: %d  current money: %d\n                    current LED: %b    current
value: %b ",
distance_out, money, scan, display);

endmodule
```

The period of **clk** is 100ns---- 50ns high level, 50ns low level.

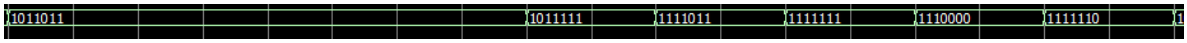
The period of **clk2** is 4ns---- 2ns high level, 2ns low level.

At first I assign the value of **distance_out** is 12345, the value of **money** is 6666.

After 200ns, their value change to 78965 and 5555 respectively.

The reason why I assign each bit of 'money' the same is because when I check the wave form figure, it is convenient for me to distinguish whether LEDs is showing the value of distance or showing the value of money.

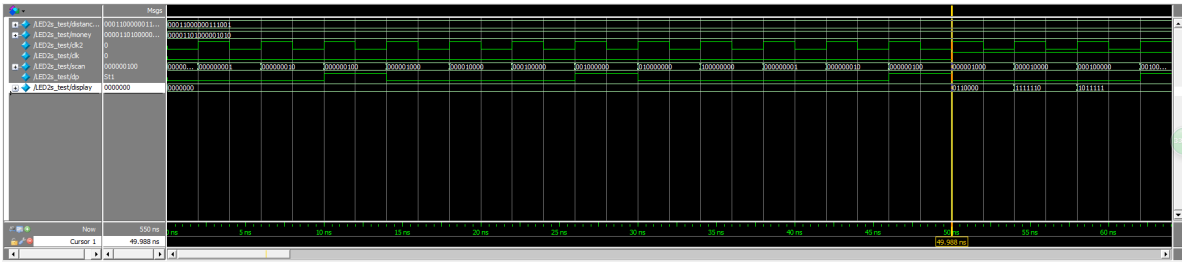
For example:



The first part is showing the money, the following parts are showing the distance.

Monitor is used to display the value of each variable in script which will be more clear, because the figure is too long to read all values at the same time.

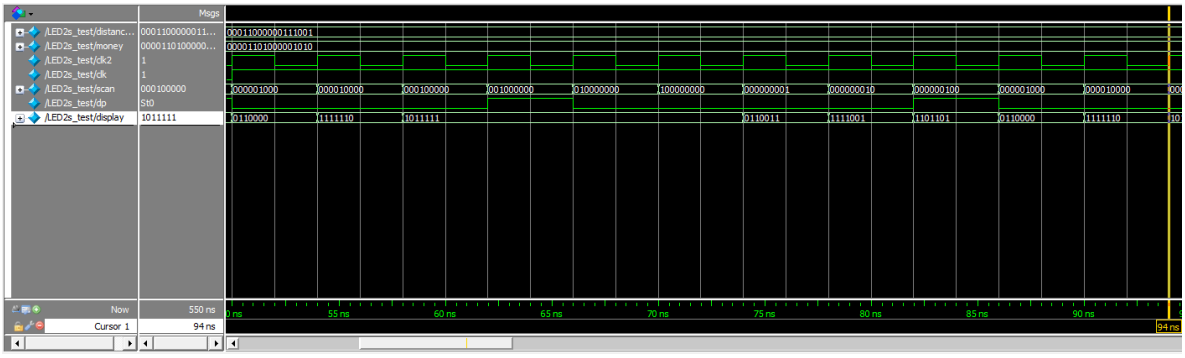
I will plot the wave form figure and results monitor showed:



```
#                  0current distance:  12345  current money:   6666
#                    current LED: 000000000   current value: 0000000
#                  2current distance:  12345  current money:   6666
#                    current LED: 000000001   current value: 0000000
#                  6current distance:  12345  current money:   6666
#                    current LED: 000000010   current value: 0000000
#                 10current distance:  12345  current money:   6666
#                    current LED: 000000100   current value: 0000000
#                 14current distance:  12345  current money:   6666
#                    current LED: 000001000   current value: 0000000
#                 18current distance:  12345  current money:   6666
#                    current LED: 000010000   current value: 0000000
#                 22current distance:  12345  current money:   6666
#                    current LED: 000100000   current value: 0000000
#                 26current distance:  12345  current money:   6666
#                    current LED: 001000000   current value: 0000000
#                 30current distance:  12345  current money:   6666
#                    current LED: 010000000   current value: 0000000
#                 34current distance:  12345  current money:   6666
#                    current LED: 100000000   current value: 0000000
#                 38current distance:  12345  current money:   6666
#                    current LED: 000000001   current value: 0000000
#                 42current distance:  12345  current money:   6666
#                    current LED: 000000010   current value: 0000000
#                 46current distance:  12345  current money:   6666
#                    current LED: 000000100   current value: 0000000
```

As you can see, at first the distance is 12345, money is 6666, but each of the LED doesn't show any value, this is because clk2 doesn't meet posedge at that time, so the value of variables---'**X1_distance**', '**X2_distance**'and so on are "X"(undetermined), LEDs will display BLANK.

After 50ns, variables hold values, figures as shown below:



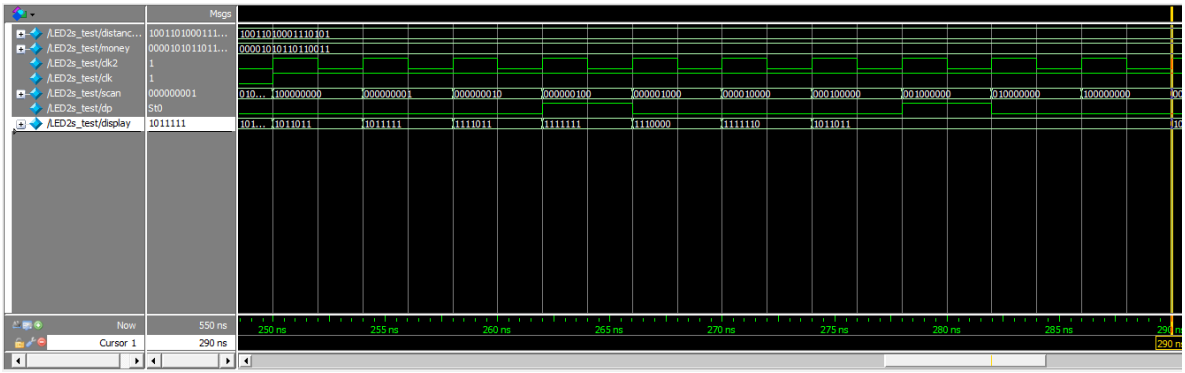
```
50current distance:  12345  current money:   6666
  current LED: 000001000    current value: 0110000
54current distance:  12345  current money:   6666
  current LED: 000010000    current value: 1111110
58current distance:  12345  current money:   6666
  current LED: 000100000    current value: 1011111
62current distance:  12345  current money:   6666
  current LED: 001000000    current value: 1011111
66current distance:  12345  current money:   6666
  current LED: 010000000    current value: 1011111
70current distance:  12345  current money:   6666
  current LED: 100000000    current value: 1011111
74current distance:  12345  current money:   6666
  current LED: 000000001    current value: 0110011
78current distance:  12345  current money:   6666
  current LED: 000000010    current value: 1111001
82current distance:  12345  current money:   6666
  current LED: 000000100    current value: 1101101
86current distance:  12345  current money:   6666
  current LED: 000001000    current value: 0110000
90current distance:  12345  current money:   6666
  current LED: 000010000    current value: 1111110
```

As we expected, all of the LEDs turn on and off one by one, showing the value of distance and money bit by bit with the correct number.

After 200ns, the value of distance and money change to 78965 and 5555, however, at that time the '**clk**' is at low level, thus the value of each intermediate such as '**X1_distance**' will not change. The figure as shown below:

```
198current distance:  12345  current money:   6666
   current LED: 000010000    current value: 1111110
200current distance:  78965  current money:   5555
   current LED: 000010000    current value: 1111110
202current distance:  78965  current money:   5555
   current LED: 000100000    current value: 1011111
206current distance:  78965  current money:   5555
   current LED: 001000000    current value: 1011111
210current distance:  78965  current money:   5555
   current LED: 010000000    current value: 1011111
214current distance:  78965  current money:   5555
   current LED: 100000000    current value: 1011111
218current distance:  78965  current money:   5555
   current LED: 000000001    current value: 0110011
222current distance:  78965  current money:   5555
   current LED: 000000010    current value: 1111001
226current distance:  78965  current money:   5555
   current LED: 000000100    current value: 1101101
230current distance:  78965  current money:   5555
   current LED: 000001000    current value: 0110000
234current distance:  78965  current money:   5555
   current LED: 000010000    current value: 1111110
238current distance:  78965  current money:   5555
   current LED: 000100000    current value: 1011111
242current distance:  78965  current money:   5555
   current LED: 001000000    current value: 1011111
246current distance:  78965  current money:   5555
   current LED: 010000000    current value: 1011111
250current distance:  78965  current money:   5555
   current LED: 100000000    current value: 1011011
```

After 250ns, the value of each intermediate variables change, and we get the correct answer:




```
246current distance:  78965  current money:   5555
   current LED: 010000000    current value: 1011111
250current distance:  78965  current money:   5555
   current LED: 100000000    current value: 1011011
254current distance:  78965  current money:   5555
   current LED: 000000001    current value: 1011111
258current distance:  78965  current money:   5555
   current LED: 000000010    current value: 1111011
262current distance:  78965  current money:   5555
   current LED: 000000100    current value: 1111111
266current distance:  78965  current money:   5555
   current LED: 000001000    current value: 1110000
270current distance:  78965  current money:   5555
   current LED: 000010000    current value: 1111110
274current distance:  78965  current money:   5555
   current LED: 000100000    current value: 1011011
278current distance:  78965  current money:   5555
   current LED: 001000000    current value: 1011011
282current distance:  78965  current money:   5555
   current LED: 010000000    current value: 1011011
286current distance:  78965  current money:   5555
   current LED: 100000000    current value: 1011011
```