一、设计任务

本次课程设计需要完成一个地铁售票机，用于模仿地铁售票的自动售票，完成地铁售票的核心控制功能。

地铁售票机有两个进币孔，可以输入硬币和纸币，售货机有两个进币孔，一个是输入硬币，一个是输入纸币，硬币的识别范围是1 元的硬币，纸币的识别范围是5 元，10 元，20元。乘客可以连续多次投入钱币。

地铁以南京市轨道交通1/2/3/4号线为基准进行设计考虑。站点数较多，需自行编码。系统可以通过按键设定当前站点为4条线路中任意一站。

乘客买票时可以有两种选择，第一种，乘客已经知道所需费用，直接选择票价，如2元、3元或4元或更多。第二种，不知道票价，选择出站口，系统以目的地与当前站的站数来进行计算价格，计算方式参考南京市轨道交通计价标准。请注意，由于换乘站的存在导致两地之间有可能有多种价格的，以最低价格为准。得到票价单价后，乘客可以选择所需购买的票数，然后进行投币，投入的钱币达到所需金额时，售票机自动出票，并一次性找出余额，本次交易结束，等待下一次的交易。在投币期间，乘客可以按取消键取消本次操作，钱币自动一次性退出。

二、设计过程

（一）方案论证

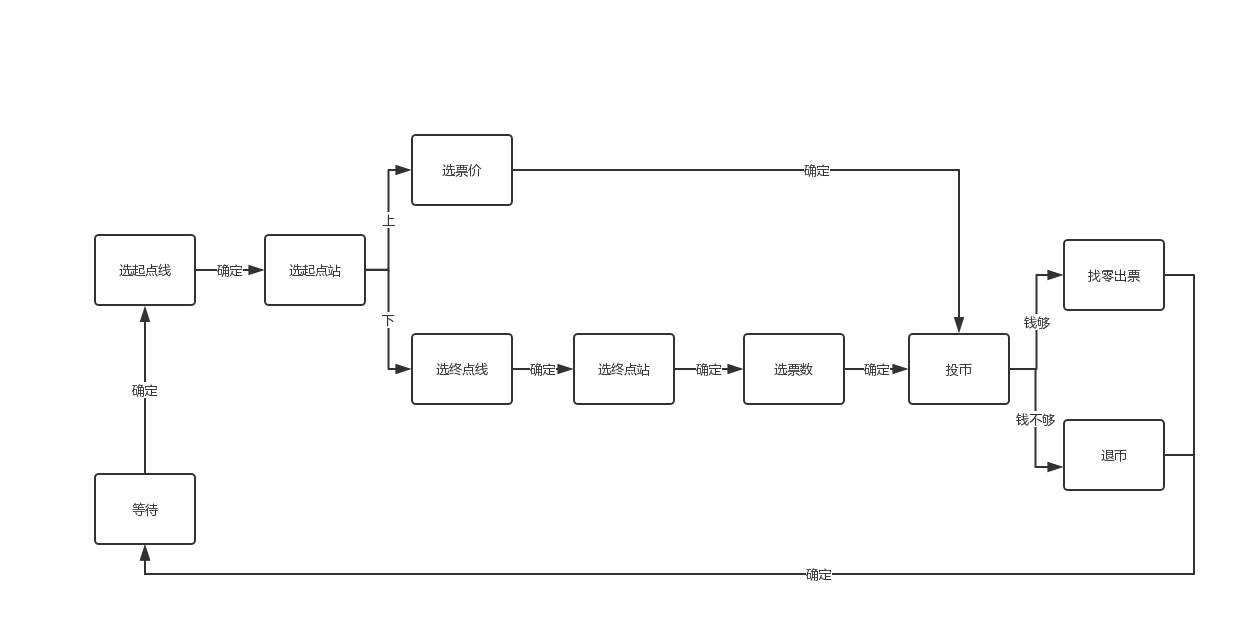
该地铁售票系统可以实现由等待进入选地铁线路状态，接着进入选当前线路地铁站点的状态。在下一个状态，我们选择是先选票价，还是先选终点站。如果选择先选终点站，则接着进入选择终点站线路，之后选择终点站的站点，选完站点后会显示这段路程票价。如果选择先选票价，就可以选择2，3，5，7元的票价。先选终点站和先选票价，最终都会进入投币状态。投完币后，如果钱够就找零出票，如果不够就退币。

其中的一个难点就是如何计算票价，我们选择的方法是将一张价格表写入FPGA的ROM中。

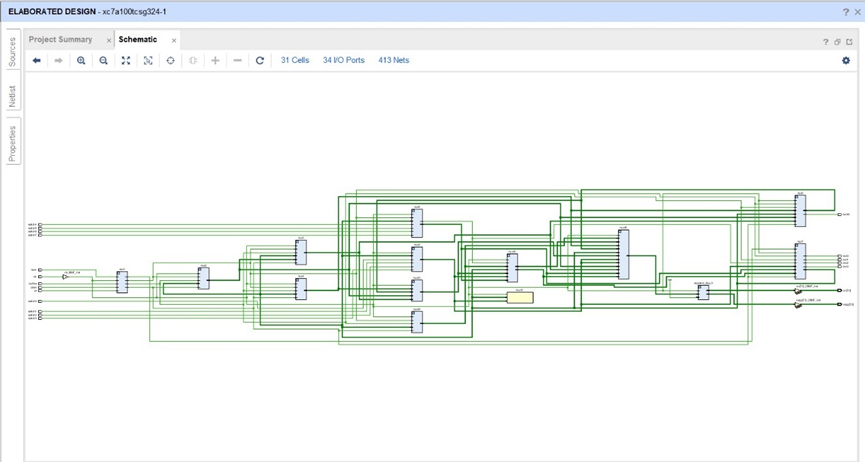
（二）系统结构图

1.电路原理图

2. 状态机



2.原理图



（三）设计思路与方法

先写思路方法，再放代码

1.顶层模块

顶层模块Ceiling只用于布置所有模块的端口映射关系，顶层模块本身实体定义中，端口是包括整个系统的输入和输出端口，即switch0、switch1、switch2、switch3、switch4、switch5、switch6、switch7、segg和an。它们分别对应着投币1元、投币5元、投币10元、投币20元、选票2元、选票3元、选票5元、选票7元、七段显示数码管的和七段显示数码管的。除了顶层模块之外的其余各模块的输入输出端口总和将顶层模块的输入输出端口完全包括，且顶层模块中定义了许多传递的信号，用以从一个子模块映射到另一个子模块中，这样某些量就可以在不同的模块中被调用。

首先，所有子模块都需要被声明，这里包括了一个七段显示数码管的IP核，声明即说明其所有端口。其次，所有子模块需要被例化，例化过程即子模块所有端口与顶层模块的端口或信号之间的对应关系。

【代码】

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity ceiling is

Port (clk,up,down,confirm,back:in std\_logic;

switch0:IN std\_logic; --四个开关，投入1元、5元、10元、20元

switch1:IN std\_logic;

switch2:IN std\_logic;

switch3:IN std\_logic;

switch4,switch5,switch6,switch7:in std\_logic; --这四个开关分别对应着快速选票价：2元、3元、5元、7元

segg :out std\_logic\_vector(7 downto 0);

an :out std\_logic\_vector(7 downto 0)

); --四个开关，投入1元、5元、10元、20元 );

end ceiling;

architecture Behavioral of ceiling is

--模块1 按键消抖

component no\_buffeting

Port (clk,up,down,confirm,back:in std\_logic;

up0,down0,confirm0,back0:out std\_logic

);--不带0的都是原按键信号,带0的是消抖后的信号

end component;

--模块2 选起始线路

component choose\_starting\_line

Port (clk,up,down,confirm:in std\_logic;

get\_present\_state:in std\_logic\_vector(3 downto 0);

starting\_line:out std\_logic\_vector(31 downto 0)

);

end component;

--模块3 选起始站点

component choose\_starting\_point

Port (clk1,up1,down1,confirm1:in std\_logic;

get\_starting\_line:in std\_logic\_vector(31 downto 0);

starting\_point:out std\_logic\_vector(31 downto 0);

get\_present\_state:in std\_logic\_vector(3 downto 0)

);

end component;

--模块4 选终点线路

component choose\_end\_line

Port (clk,up,down,confirm:in std\_logic;

end\_line:out std\_logic\_vector(31 downto 0);

get\_present\_state:in std\_logic\_vector(3 downto 0)

);

end component;

--模块5 选终点站点

component choose\_end\_point

Port (clk2,up2,down2,confirm2:in std\_logic;

led15:out std\_logic;

get\_starting\_line,get\_end\_line:in std\_logic\_vector(31 downto 0);

get\_starting\_point:in std\_logic\_vector(31 downto 0);

end\_point:out std\_logic\_vector(31 downto 0);

get\_present\_state:in std\_logic\_vector(3 downto 0)

);

end component;

--模块7 状态机

component top\_entity

Port(

clk:IN std\_logic; --时钟信号

confirm:IN std\_logic; --按钮"确定"

back:IN std\_logic; --按钮"返回"

up:IN std\_logic; --按钮"加一"

down:IN std\_logic; --按钮"减一"

get\_total,get\_ticket\_price:IN std\_logic\_vector(31 downto 0);

get\_real\_pay:IN std\_logic\_vector(31 downto 0);

get\_up\_to\_5sec,get\_flag0:IN std\_logic;

status:out std\_logic\_vector(3 downto 0)

);

end component;

--模块8 选票价

component choose\_price

Port (clk,switch4,switch5,switch6,switch7,confirm:in std\_logic; --这四个开关分别对应着快速选票价：2元、3元、5元、7元

ticket\_price:out std\_logic\_vector(31 downto 0);

flag0:out std\_logic;

get\_present\_state:in std\_logic\_vector(3 downto 0)

);

end component;

--模块9 投币

component pay

Port (clk,switch0,switch1,switch2,switch3:in std\_logic;

real\_pay:out std\_logic\_vector(31 downto 0);

get\_present\_state:in std\_logic\_vector(3 downto 0)

);

end component;

--模块10 找零出票

component computechange

Port (clk,flag0:in std\_logic;

get\_amount:in std\_logic\_vector(31 downto 0);

get\_real\_pay:in std\_logic\_vector(31 downto 0);

change,total:out std\_logic\_vector(31 downto 0);

get\_price,ticket\_price:in std\_logic\_vector(31 downto 0);

get\_present\_state:in std\_logic\_vector(3 downto 0));

end component;

--模块11 退币

component refund

Port (clk:in std\_logic;

get\_real\_pay:in std\_logic\_vector(31 downto 0);

get\_present\_state:in std\_logic\_vector(3 downto 0)

);

end component;

--模块12 数码管

component seven\_segment\_disp\_0

port

(

clk :in std\_logic;

dispdata :in std\_logic\_vector(31 downto 0);

seg\_able :in std\_logic\_vector(7 downto 0);

segg :out std\_logic\_vector(7 downto 0);

an :out std\_logic\_vector(7 downto 0)

);

end component;

--模块13 计算票价

component dispram

port

(

clk :in std\_logic;

startline,endline:in std\_logic\_vector(31 downto 0);

startpoint,endpoint:in std\_logic\_vector(31 downto 0);

price:out std\_logic\_vector(31 downto 0)

);

end component;

--模块14 选票数

component amount

Port (clk,up,down,confirm:in std\_logic;

ticket\_amount:out std\_logic\_vector(31 downto 0);

get\_present\_state:in std\_logic\_vector(3 downto 0);

int\_amount:out integer

);

end component;

--模块15 显示模块

component display

Port (clk:in std\_logic;

get\_present\_state:in std\_logic\_vector(3 downto 0);

dispdata :out std\_logic\_vector(31 downto 0 );

get\_starting\_line:in std\_logic\_vector(31 downto 0 );

get\_starting\_point:in std\_logic\_vector(31 downto 0);

get\_end\_line: in std\_logic\_vector(31 downto 0 );

get\_end\_point :in std\_logic\_vector(31 downto 0 );

get\_ticket\_price:in std\_logic\_vector(31 downto 0 );

get\_real\_pay:in std\_logic\_vector(31 downto 0 );

get\_change:in std\_logic\_vector(31 downto 0 );

get\_refund:in std\_logic\_vector(31 downto 0 );

get\_amount:in std\_logic\_vector(31 downto 0 );

get\_price:in std\_logic\_vector(31 downto 0 );

get\_total:in std\_logic\_vector(31 downto 0 )

);

end component;

signal disp\_data\_2 : std\_logic\_vector(31 downto 0);

signal seg\_able\_2 : std\_logic\_vector( 7 downto 0);

signal sig\_up0,sig\_down0,sig\_confirm0,sig\_back0:std\_logic;

signal int\_amount:integer;

signal starting\_line,end\_line: std\_logic\_vector(31 downto 0);

signal total,change:std\_logic\_vector(31 downto 0);

signal starting\_point,end\_point:std\_logic\_vector(31 downto 0);

signal up\_to\_5sec,flag0:std\_logic;

signal present\_state:std\_logic\_vector(3 downto 0);

signal price2:std\_logic\_vector(31 downto 0); --票价表里的票价

signal ticket\_amount:std\_logic\_vector(31 downto 0);

signal ticket\_price:std\_logic\_vector(31 downto 0);

signal real\_pay:std\_logic\_vector(31 downto 0);

begin

mux1:no\_buffeting

port map(

clk=>clk,

up=>up,

down=>down,

confirm=>confirm,

back=>back,

up0=>sig\_up0,

down0=>sig\_down0,

confirm0=>sig\_confirm0,

back0=>sig\_back0

);

mux2:choose\_starting\_line

port map(

clk=>clk,

up=>sig\_up0,

down=>sig\_down0,

confirm=>sig\_confirm0,

starting\_line=>starting\_line,

get\_present\_state=>present\_state

);

mux3:choose\_starting\_point

port map(

clk1=>clk,

up1=>sig\_up0,

down1=>sig\_down0,

confirm1=>sig\_confirm0,

get\_starting\_line=>starting\_line,

starting\_point=>starting\_point,

get\_present\_state=>present\_state

);

mux4:choose\_end\_line

port map(

clk=>clk,

up=>sig\_up0,

down=>sig\_down0,

confirm=>sig\_confirm0,

end\_line=>end\_line,

get\_present\_state=>present\_state

);

mux5:choose\_end\_point

port map(

clk2=>clk,

led15=>led15,

up2=>sig\_up0,

down2=>sig\_down0,

confirm2=>sig\_confirm0,

get\_starting\_line=>starting\_line,

get\_starting\_point=>starting\_point,

get\_end\_line=>end\_line,

end\_point=>end\_point,

get\_present\_state=>present\_state

);

mux7:top\_entity

port map(

clk=>clk,

up=>sig\_up0,

down=>sig\_down0,

confirm=>sig\_confirm0,

back=>sig\_back0,

get\_total=>total,

get\_ticket\_price=>ticket\_price,

get\_real\_pay=>real\_pay,

get\_up\_to\_5sec=>up\_to\_5sec,

get\_flag0=>flag0,

status=>present\_state

);

mux8:choose\_price

port map(

clk=>clk,

confirm=>sig\_confirm0,

switch4=>switch4,

switch5=>switch5,

switch6=>switch6,

switch7=>switch7,

flag0=>flag0,

ticket\_price=>ticket\_price,

get\_present\_state=>present\_state

);

mux9:pay

port map(

clk=>clk,

switch0=>switch0,

switch1=>switch1,

switch2=>switch2,

switch3=>switch3,

real\_pay=>real\_pay,

get\_present\_state=>present\_state

);

mux10:computechange

port map(

clk=>clk,

flag0=>flag0,

ticket\_price=>ticket\_price,

get\_real\_pay=>real\_pay,

change=>change,

total=>total,

get\_price=>price2, --票价表里的票价

get\_amount=>ticket\_amount,

get\_present\_state=>present\_state

);

mux11:refund

port map(

clk=>clk,

get\_real\_pay=>real\_pay,

get\_present\_state=>present\_state

);

seg\_able\_2 <= "11111111";

segment\_disp\_0 : seven\_segment\_disp\_0

port map(

clk => clk,

dispdata => disp\_data\_2,

seg\_able => seg\_able\_2,

segg => segg,

an => an

);

mux13:dispram

port map(

clk=>clk,

startline=>starting\_line,

startpoint=>starting\_point,

endline=>end\_line,

endpoint=>end\_point,

price=>price2

);

mux14:amount

port map(

clk=>clk,

up=>sig\_up0,

down=>sig\_down0,

confirm=>sig\_confirm0,

ticket\_amount=>ticket\_amount,

get\_present\_state=>present\_state,

int\_amount=>int\_amount

);

mux15:display

port map(

clk=>clk,

get\_present\_state=>present\_state,

dispdata =>disp\_data\_2,

get\_starting\_line=>starting\_line,

get\_starting\_point=>starting\_point,

get\_end\_line=>end\_line,

get\_end\_point =>end\_point,

get\_ticket\_price=>ticket\_price,

get\_real\_pay=>real\_pay,

get\_change=>change,

get\_refund=>real\_pay,

get\_amount=>ticket\_amount,

get\_price=>price2,

get\_total=>total

);

end Behavioral;

2.按键消抖模块

为了确保程序对按键的一次闭合或者一次断开只响应一次,必须进行按键的消抖处理。按键消抖中，输入信号为时钟信号和四个按键信号（上、下、确认、返回），输出信号为消抖后的四个按键信号。

由于要对四个按键进行消抖，故选取四个并行的进程对它们分别消抖。以对“上”键进行消抖为例，首先定义一个计数变量count为0~500000的整数，因为我们使用的板子本身的时钟频率为100MHz，而按键是按照一个脉冲计算，一个脉冲约5ms，对应count计数即为500000。消抖原理即，按键行为只有达到一个脉冲的时长才会被视为真正按下，否则被视为毛刺信号。若检测到有按键按下，在时钟扫描过程中，若按下时间计数达到499999，则令count加一，否则count保持不变，再对count进行判断，若count达到500000及以上，则确定按键按下符合要求，否则按键无效。

【代码】

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity no\_buffeting is

Port (clk,up,down,confirm,back:in std\_logic;

up0,down0,confirm0,back0:out std\_logic);--不带0的都是原按键信号,带0的是消抖后的信号

end no\_buffeting;

architecture Behavioral of no\_buffeting is

begin

up\_no\_buffeting:process(clk,up)

variable count:integer range 0 to 500000; --1MHz对应1us，100MHz对应0.01us，按键是1一个脉冲，一个脉冲约5ms，也就是100000\*5=500000

begin

if (up='0') then

if (clk'event and clk='1') then

if count<500000 then count:=count+1;

else count:=count;

end if;

if count<=499999 then up0<='0';

else up0<='1';

end if;

end if;

else count:=0;up0<='0';

end if;

end process up\_no\_buffeting;

down\_no\_buffeting:process(clk,down)

variable count:integer range 0 to 500000; --1MHz对应1us，100MHz对应0.01us，按键是1一个脉冲，一个脉冲约5ms，也就是100000\*5=500000

begin

if (down='0') then

if (clk'event and clk='1') then

if count<500000 then count:=count+1;

else count:=count;

end if;

if count<=499999 then down0<='0';

else down0<='1';

end if;

end if;

else count:=0;down0<='0';

end if;

end process down\_no\_buffeting;

confirm\_no\_buffeting:process(clk,confirm)

variable count:integer range 0 to 500000; --1MHz对应1us，100MHz对应0.01us，按键是1一个脉冲，一个脉冲约5ms，也就是100000\*5=500000

begin

if (confirm='0') then

if (clk'event and clk='1') then

if count<500000 then count:=count+1;

else count:=count;

end if;

if count<=499999 then confirm0<='0';

else confirm0<='1';

end if;

end if;

else count:=0;confirm0<='0';

end if;

end process confirm\_no\_buffeting;

back\_no\_buffeting:process(clk,back)

variable count:integer range 0 to 500000; --1MHz对应1us，100MHz对应0.01us，按键是1一个脉冲，一个脉冲约5ms，也就是100000\*5=500000

begin

if (back='0') then

if (clk'event and clk='1') then

if count<500000 then count:=count+1;

else count:=count;

end if;

if count<=499999 then back0<='0';

else back0<='1';

end if;

end if;

else count:=0;back0<='0';

end if;

end process back\_no\_buffeting;

--四个按键分别消抖

end Behavioral;

3.选择起始线路模块

根据题目要求，本地铁售票机按照南京一、二、三、四号线的规格设置线路站点和票价，则起始线路只有四种选择，这里令0、1、2、3分别代表地铁的四条线。本模块的输入端口包括时钟信号、当前状态和上、下、确定三个按键，输出端口即起始线路（32位二进制数）。此外，我们还定义了一个32位进制起始线路信号starting\_line32和一个整数起始线路信号sig\_starting\_line，以及三个按键对应的信号up0、down0、confirm0。本模块中共有两个进程，第一个进程用于在每一个时钟上升沿到来时，将sig\_starting\_line转换为32位二进制数赋给starting\_line32（需要使用使用函数conv\_std\_logic\_vector(m,n)，且需要在文件开头声明use IEEE.STD\_LOGIC\_ARITH.ALL），再将starting\_line32赋给输出端口，将按键输入赋给对应的信号。其中，按键与信号的赋值是为了在后面判断一个按键是否真正被按下并释放，如up='1'and up0='0'表示“上”键被按一下的完整过程。在第二个进程中，判断当前处于0000（等待状态）的情况下，起始线路为0，这样等价于每次回到等待状态上一次的操作都会清零；当前处于0001（选择起始线路）的情况下，通过操作“上”和“下”键加减线路值，线路会实时传递给输出端口。

【代码】

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;--使用函数conv\_std\_logic\_vector(m,n)的前提

entity choose\_starting\_line is

Port (clk,up,down,confirm:in std\_logic;

get\_present\_state:in std\_logic\_vector(3 downto 0);

starting\_line:out std\_logic\_vector(31 downto 0)

);

end choose\_starting\_line;

architecture Behavioral of choose\_starting\_line is

signal sig\_starting\_line:integer range 3 downto 0;--把信号sig\_starting\_line定义成一个0~3的整数

signal starting\_line32:std\_logic\_vector(31 downto 0);

signal confirm0,up0,down0:std\_logic;

begin

process(clk)

begin

if (clk'event and clk='1') then

starting\_line32<=conv\_std\_logic\_vector(sig\_starting\_line,32);

starting\_line<=starting\_line32;

confirm0<=confirm;

up0<=up;

down0<=down;

end if;

end process;

choosing:process(clk,up,down,confirm,get\_present\_state)

variable temp:integer range 3 downto 0:=0;--把变量starting\_line定义成一个1~4的整数，定义变量主要是为了在进程中实时更新线路数值

begin

if (clk'event and clk='1') then

if (get\_present\_state="0000") then

temp:=0;sig\_starting\_line<=temp;

end if;

if (get\_present\_state="0001") then

if (up='1'and up0='0') then temp:=temp+1;sig\_starting\_line<=temp;end if;

if (down='1'and down0='0') then temp:=temp-1;sig\_starting\_line<=temp;end if;

end if;

end if;

end process choosing;

end Behavioral;

4.选择起始站点模块

本模块输入端口包括时钟信号、上、下、确认按键、当前状态和起始线路，输出端口即起始站点。原现将起始线路也设置为输入信号，因为这四条线路的每一条线上的站点数都不同，我们希望通过判断起始线路限定每条线的站点数量，但是后来由于实际问题还是删去了这一判断过程，但端口仍保留在实体中。

接着，我们定义了三个对应的按键信号、整数起始站点信号sig\_starting\_point和32位二进制起始站点信号starting\_point32，第一个进程与选择起始线路模块的第一个进程一致，都是在每一个时钟上升沿到来的时候为端口或信号赋值。在第二个进程中，判断在0000（等待状态）的情况下，起始站点为0；当前处于0010（选择起始站点）的情况下，通过操作“上”和“下”键加减站点值，站点会实时传递给输出端口。

【代码】

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity choose\_starting\_point is

Port (clk1,up1,down1,confirm1:in std\_logic;

get\_starting\_line:in std\_logic\_vector(31 downto 0);

starting\_point:out std\_logic\_vector(31 downto 0);

get\_present\_state:in std\_logic\_vector(3 downto 0)

);

end choose\_starting\_point;

signal confirm10,up10,down10:std\_logic;

signal sig\_starting\_point:integer;

signal starting\_point32:std\_logic\_vector(31 downto 0);

begin

process(clk1)

begin

if (clk1'event and clk1='1') then

starting\_point32<=conv\_std\_logic\_vector(sig\_starting\_point,32);

starting\_point<=starting\_point32;

confirm10<=confirm1;

up10<=up1;

down10<=down1;

end if;

end process;

choosing:process(clk1,up1,down1,confirm1,get\_present\_state)

variable temp1:integer range 28 downto 0:=0;

begin

if (clk1'event and clk1='1') then

if (get\_present\_state="0000") then

temp1:=0;sig\_starting\_point<=temp1;

end if;

if (get\_present\_state="0010") then

if (up1='1'and up10='0') then temp1:=temp1+1;sig\_starting\_point<=temp1;end if;

if (down1='1'and down10='0') then temp1:=temp1-1;sig\_starting\_point<=temp1;end if;

--if (confirm1='1'and confirm10='0') then sig\_starting\_point<=temp1;end if;

--starting\_point<=conv\_std\_logic\_vector(sig\_starting\_1\_point,5);--把整数站点转换成5位二进制数

end if;

end if;

end process choosing;

end Behavioral;

5.选择终点线路模块

此模块与选择起始线路模块结构基本一致。

【代码】

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;--使用函数conv\_std\_logic\_vector(m,n)的前提

entity choose\_end\_line is

Port (clk,up,down,confirm:in std\_logic;

end\_line:out std\_logic\_vector(31 downto 0);

get\_present\_state:in std\_logic\_vector(3 downto 0)

);

end choose\_end\_line;

architecture Behavioral of choose\_end\_line is

signal end\_line32:std\_logic\_vector(31 downto 0);

signal sig\_end\_line:integer range 3 downto 0;--把信号sig\_end\_line定义成一个1~4的整数

signal confirm0,up0,down0:std\_logic;

begin

process(clk)

begin

if (clk'event and clk='1') then

end\_line32<=conv\_std\_logic\_vector(sig\_end\_line,32);

end\_line<=end\_line32;

confirm0<=confirm;

up0<=up;

down0<=down;

end if;

end process;

choosing:process(clk,up,down,confirm,get\_present\_state)

variable temp:integer range 3 downto 0:=0;--把变量starting\_line定义成一个1~4的整数，定义变量主要是为了在进程中实时更新线路数值

begin

if (clk'event and clk='1') then

if (get\_present\_state="0000") then

temp:=0;sig\_end\_line<=temp;

end if;

if (get\_present\_state="0100") then

if (up='1'and up0='0') then temp:=temp+1;sig\_end\_line<=temp;end if;

if (down='1'and down0='0') then temp:=temp-1;sig\_end\_line<=temp;end if;

end if;

end if;

end process choosing;

end Behavioral;

6.选择终点站点模块

此模块与选择站点线路模块结构基本一致。

【代码】

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity choose\_end\_point is

Port (clk2,up2,down2,confirm2:in std\_logic;

led15:out std\_logic;

get\_starting\_line,get\_end\_line:in std\_logic\_vector(31 downto 0);

get\_starting\_point:in std\_logic\_vector(31 downto 0);

end\_point:out std\_logic\_vector(31 downto 0);

get\_present\_state:in std\_logic\_vector(3 downto 0)

);

end choose\_end\_point;

architecture Behavioral of choose\_end\_point is

signal end\_point32:std\_logic\_vector(31 downto 0);

signal confirm0,up0,down0:std\_logic;

signal warning:std\_logic; --警告，起始站和终点站一致，在顶层模块里可以调用它并亮灯...

signal sig\_end\_point:integer;

begin

process(clk2)

begin

if (clk2'event and clk2='1') then

end\_point32<=conv\_std\_logic\_vector(sig\_end\_point,32);

end\_point<=end\_point32;

confirm0<=confirm2;

up0<=up2;

down0<=down2;

led15<=warning;

end if;

end process;

choosing:process(clk2,up2,down2,confirm2,get\_present\_state)

variable temp1:integer range 28 downto 0:=0;

begin

if (clk2'event and clk2='1') then

if (get\_present\_state="0000") then

temp1:=0;sig\_end\_point<=temp1;

warning<='0';

end if;

if (get\_present\_state="0101") then

if (up2='1'and up0='0') then temp1:=temp1+1;sig\_end\_point<=temp1;end if;

if (down2='1'and down0='0') then temp1:=temp1-1;sig\_end\_point<=temp1;end if;

end if;

end if;

end process choosing;

end Behavioral;

7.状态机模块

本模块实现了状态的转换，在操作过程中共十二个状态，定义status为表征当前状态的输出端口，它从0000~1100均对应明确的状态。另外，输入端口中有经端口映射得到的若干输入信号，包括地铁票总价、直接选择的票价、实际付款等，这些信号在某些状态之间的转换中需要被用于判断。总的状态机转换如图所示，图中省略了每一步的返回操作，若在当前状态按下返回，则可回到上一状态修改操作。

这里以当前状态为投币状态为例，对状态的转换作详细阐释。首先，需要判断flag0，此处flag0是经端口映射后得到的一个标志位，若系统经过了直接选票价的操作，则标志位被置1，否则标志位为0。若投币状态下，标志位为1，则判断实际付款与票价的关系，若付款足够支付则跳转到找零出票状态，否则跳转到退币状态；标志位为0，则判断实际付款与总应付票价的关系，若付款足够支付则跳转到找零出票状态，否则跳转到退币状态。最后，不论是找零出票状态还是退币状态，按下“确定”都会跳转到等待状态，实现状态机的循环。

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;--可以使用STD\_LOGIC,STD\_LOGIC\_VECTOR

use IEEE.STD\_LOGIC\_ARITH.ALL;--数据类型转换

use IEEE.STD\_LOGIC\_UNSIGNED.All;--数据类型转换

entity top\_entity is

Port(

clk:IN std\_logic; --时钟信号

confirm:IN std\_logic; --按钮"确定"

back:IN std\_logic; --按钮"返回"

up:IN std\_logic; --按钮"加一"

down:IN std\_logic; --按钮"减一"

get\_total,get\_ticket\_price:IN std\_logic\_vector(31 downto 0);

get\_real\_pay:IN std\_logic\_vector(31 downto 0);

get\_up\_to\_5sec,get\_flag0:IN std\_logic;

status:out std\_logic\_vector(3 downto 0)

);

end top\_entity;

architecture Behavioral of top\_entity is

signal flag:std\_logic;

signal present\_state:std\_logic\_vector(3 downto 0);

signal confirm0,back0,up0,down0:std\_logic;

begin

process(clk)

begin

if (clk'event and clk='1') then

status<=present\_state;

confirm0<=confirm;

back0<=back;

up0<=up;

down0<=up;

end if;

end process;

process(clk,confirm,back,up,down,present\_state)

begin

if (clk'event and clk='1') then

case present\_state is

when "0000"=> --等待状态

if (confirm0='0' and confirm='1') then --按确认进入选起始线状态

present\_state<="0001";

end if;

when "0001"=> --选当前站线路状态

if (confirm0='0' and confirm='1') then present\_state<="0010";end if;

if (back0='0' and back='1') then present\_state<="0000";end if;

when "0010"=> --选当前站站点状态

if (confirm0='0' and confirm='1') then present\_state<="1010";end if; --按确定跳到状态10

if (back0='0' and back='1') then present\_state<="0001";end if;

when "0011"=> --选票价状态

if (confirm0='0' and confirm='1') then present\_state<="0111";end if; --选票价确认后直接投币

if (back0='0' and back='1') then present\_state<="0010";end if;

when"0100"=> --选目的站线路状态

if (confirm0='0' and confirm='1') then present\_state<="0101";end if;

if (back0='0' and back='1') then present\_state<="0010";

end if;

when "0101"=> --选目的站站点状态

if (confirm0='0' and confirm='1') then present\_state<="1011";end if; --按确认 去显示单价

if (back0='0' and back='1') then present\_state<="0100";end if;

when "0110"=> --选票数状态

if (confirm0='0' and confirm='1') then present\_state<="1100";end if; --按确认 去显示总价

if (back0='0' and back='1') then present\_state<="0101";end if; --按返回 去重选终点站

when "0111"=> --投币状态

if (confirm0='0' and confirm='1') then --按下确定键

if (get\_flag0='1') then

if (get\_real\_pay>=get\_ticket\_price) then present\_state<="1000";

elsif(get\_real\_pay<get\_ticket\_price) then present\_state<="1001";

end if;

elsif (get\_flag0='0') then

if (get\_real\_pay>=get\_total) then present\_state<="1000"; --如果付的钱比应付的多，就跳到找零出票状态

elsif(get\_real\_pay<get\_total) then present\_state<="1001"; --钱不够跳到退币状态

end if;

end if;

end if;

if (back0='0' and back='1') then

present\_state<="1001"; --按返回就到退币状态，即一次性退出钱币

end if;

when "1000"=> --找零出票状态

if (confirm0='0' and confirm='1') then present\_state<="0000";end if;

--if (get\_up\_to\_5sec='1') then present\_state<="0000";end if; --如果过了5秒没操作，跳到等待状态

when "1001"=> --退币状态

if (confirm0='0' and confirm='1') then present\_state<="0000";end if;

-- if (get\_up\_to\_5sec='1') then present\_state<="0000";end if; --如果过了5秒没操作，跳到等待状态

when "1010"=> --状态10 判断进入选票价还是选终点线

if (up0='0' and up='1') then present\_state<="0011";end if; --按上 去选票价

if (down0='0' and down='1') then present\_state<="0100";end if; --按下 去选终点线

if (back0='0' and back='1') then present\_state<="0010";end if;

when "1011"=> --显示起始站到终点站的单价

if (confirm0='0' and confirm='1') then present\_state<="0110";end if; --按确认 去选票数

if (back0='0' and back='1') then present\_state<="0101";end if; --按返回 去选终点站

when "1100"=> --显示总价（单价\*票数）

if (confirm0='0' and confirm='1') then present\_state<="0111";end if; --按确认 去投币

if (back0='0' and back='1') then present\_state<="0110";end if; --按返回 去重选票数

when others=> --初始状态是s1

present\_state<="0000";

end case;

end if;

end process;

end Behavioral;

8.选择票价模块

9.票价计算模块

直接将得到的起始线路和起始站点的数据以及终点线路和终点站点导入该模块，通过计算得到真实的物理地址，然后地址传入BRAM，获取数据，即票价。

此外该模块还负责将存入的照片数据通过VGA在显示屏上显示，虽然失败了

该模块使用了两个IP核 时钟IP核clocking wizard和 RAM&ROM核Block Memory Generator

其中 clocking wizard 将原100Mhz的时钟分频成31.5Mhz。此分频时钟用于VGA 640 \* 480@75 显示模式

BRAM配置如下，选择真双端口模式，AB两个端口都是12bit宽度，深度为317200。读优先，没有输出缓冲寄存器。使用真双端口模式是因为考虑到地铁票价表的读取和VGA的显示，

读优先是因为不需要写入数据。

通过python脚本编写.coe文件用于初始化BRAM。 coe文件格式如下

memory\_initialization\_radix = 16;

memory\_initialization\_vector =

2,

2,

2,

…

fff;

开头第一行 memory\_initialization\_radix = 16; 表示文件读取使用16进制。

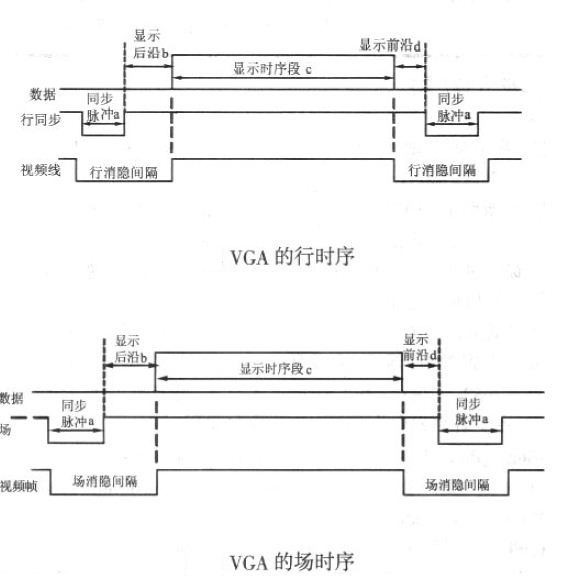
第二行下面有317200行。考虑到南京地铁一至四号线总共100站，所以其中前10000行存有地铁票价表数据。考虑到640 \* 480 = 307200，故为了通过VGA显示一张分辨率为640 \* 480 的照片，coe文件的剩下307200行存照片的像素数据。

为了写入coe文件，我使用了python3.74和numpy和opencv-python库。先通过office365中的excel打开电子表格，并将其转换成csv格式。之后用python读取csv文件将其写入txt文件中。

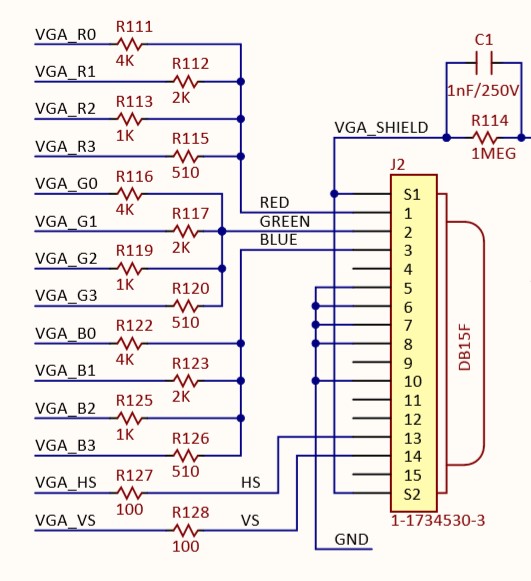
这里补充说明一下我们没有成功显示的VGA的原理。显示器扫描方式分为逐行扫描和隔行扫描：逐行扫描是扫描从屏幕左上角一点开始，从左像右逐点扫描，每扫描完一行,电子束回到屏幕的左边下一行的起始位置，在这期间，CRT对电子束进行消隐，每行结束时，用行同步信号进行同步；当扫描完所有的行，形成一帧，用场同步信号进行场同步，并使扫描回到屏幕左上方，同时进行场消隐,开始下一帧。

　　完成一行扫描的时间称为水平扫描时间，其倒数称为行频率；完成一帧（整屏）扫描的时间称为垂直扫描时间，其倒数称为场频率，即刷新一屏的频率，常见的有60Hz，75Hz等等。标准的VGA显示的场频60Hz,行频31.5KHz。

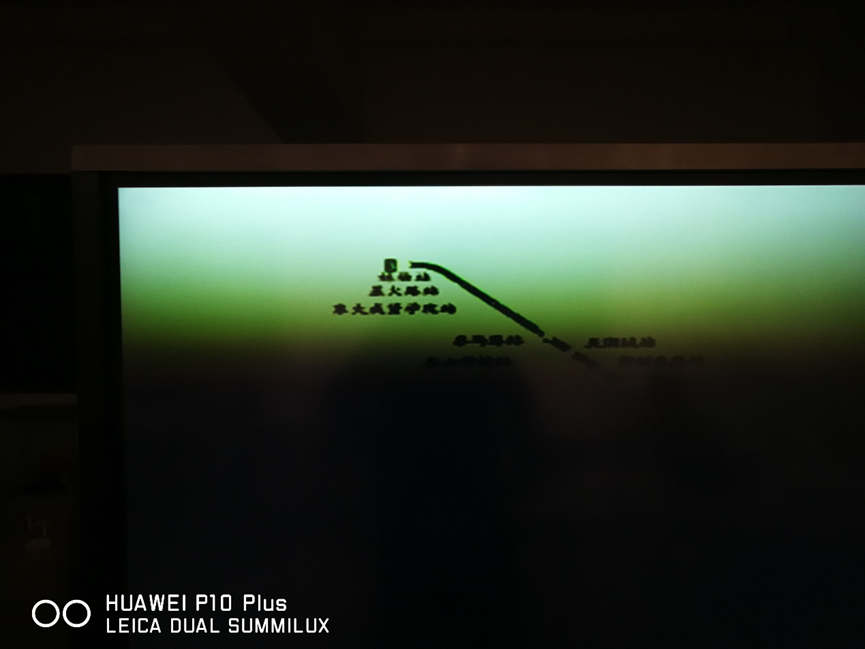
　　行场消隐信号:是针对老式显像管的成像扫描电路而言的。电子枪所发出的电子束从屏幕的左上角开始向右扫描，一行扫完需将电子束从右边移回到左边以便扫描第二行。在移动期间就必须有一个信号加到电路上，使得电子束不能发出。不然这个回扫线会破坏屏幕图像的。这个阻止回扫线产生的信号就叫作消隐信号,场信号的消隐也是一个道理。



我们Nexys4 DDR的VGA接口要输入红绿蓝三色模拟信号和行同步信号以及场同步信号，每一色输入都有4bit的位宽共计12位宽度。为了显示640\*480的照片 为此我们使用了ROM&RAM，配置其为12bit宽度，我们将307200个像素存在价格表之后。我们编写python脚本将照片颜色从24bit分辨率变换成12bit分辨率并按格式续写在coe文件之中。



最后显示的时候有问题，可以看出图片上面部分可以显示，但是下面显示的不正常。所以最后我们只好将VGA显示模块注释掉了。



【代码】

module dispram(

input clk,

input[31:0]startline,

input[31:0]startpoint,

input[31:0]endline,

input[31:0]endpoint,

output[31:0]price

);

wire clk\_disp;

wire[18:0]ramaddra ;

wire[11:0]ramdouta ;

reg[18:0]ramaddrb = 19'd10000;

wire[11:0]ramdoutb ;

begin

clk\_wiz\_0 uut\_tim(

.clk\_in1(clk),

.clk\_out1(clk\_disp)

);

reg[18:0]romaddra\_host = 19'h0;

reg[18:0]romaddra\_slave = 19'h0;

reg[6:0]set\_100 = 7'd100;

reg[18:0]price\_addr;

assign price[3:0] = ramdouta[3:0];

always @(negedge clk)

begin

case(startline)

2'b00:

begin

romaddra\_host = (startpoint + 0 )\*set\_100;

end

2'b01:

begin

romaddra\_host = (startpoint + 27 )\*set\_100;

end

2'b10:

begin

romaddra\_host = (startpoint + 53 )\*set\_100;

end

2'b11:

begin

romaddra\_host = (startpoint + 82 )\*set\_100;

end

endcase;

case(endline)

2'b00:

begin

romaddra\_slave = (endpoint + 0 );

end

2'b01:

begin

romaddra\_slave = (endpoint + 27 );

end

2'b10:

begin

romaddra\_slave = (endpoint + 53 );

end

2'b11:

begin

romaddra\_slave = (endpoint + 82 );

end

endcase;

price\_addr = romaddra\_slave + romaddra\_host;

end

//computeprice cp(

//.clk(clk\_disp),

//.startline(startline),

//.startpoint(startpoint),

//.endline(endline),

//.endpoint(endpoint),

//.price\_addr(ramaddra)

//);

//VGA vga(

//.clk\_vga(clk\_disp),

//.colour(ramdoutb),

// output hsync,

// output vsync,

// output[3:0]vgared,

// output[3:0]vgagreen,

// output[3:0]vgablue,

// output[18:0]addrb

// );

blk\_mem\_gen\_0 uut\_ram(

.clka(clk\_disp),

.ena(1),

.wea(0),

.addra(price\_addr),

.dina(0),

.douta(ramdouta),

.clkb(clk),

.enb(1),

.web(0),

.addrb(ramaddrb),

.dinb(0),

.doutb(ramdoutb)

);

end

endmodule

10.投币模块

本模块输入端口包括时钟信号、四个投币开关和当前状态，输出端口即实际付款值。同前面的模块一样，这里需要定义多个信号，用于在时钟上升沿到来时相应的赋值。在主进程中，则判断若当前状态为0000（等待状态），实际付款为0；若当前状态为0111（投币状态），实际付款是开关拨动对应的币值的累加，其中拨动switch0表示投币1元，拨动switch1表示投币5元，拨动switch2表示投币10元，拨动switch3表示投币20元。通过变量和信号的布置，投币可以连续累加并实时传递给输出端口。

【代码】

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;--使用函数conv\_std\_logic\_vector(m,n)的前提

entity pay is

Port (clk,switch0,switch1,switch2,switch3,confirm:in std\_logic;

real\_pay:out std\_logic\_vector(31 downto 0);

get\_present\_state:in std\_logic\_vector(3 downto 0)

);

end pay;

architecture Behavioral of pay is

signal sig\_pay:integer;

signal sig\_pay32:std\_logic\_vector(31 downto 0);

signal confirm0,switch00,switch10,switch20,switch30:std\_logic;

begin

process(clk)

begin

if (clk'event and clk='1') then

sig\_pay32<=conv\_std\_logic\_vector(sig\_pay,32);

real\_pay<=sig\_pay32;

confirm0<=confirm;

switch00<=switch0;

switch10<=switch1;

switch20<=switch2;

switch30<=switch3;

end if;

end process;

process(clk,confirm,switch0,switch1,switch2,switch3,get\_present\_state)

variable temp:integer:=0;

begin

if (clk'event and clk='1') then

if (get\_present\_state="0000") then

temp:=0;sig\_pay<=temp;

end if;

if (get\_present\_state="0111") then

if (switch0='1'and switch00='0') then temp:=temp+1;sig\_pay<=temp;end if; --拨动第一个开关上下一次算投入1元，下同

if (switch1='1'and switch10='0') then temp:=temp+5;sig\_pay<=temp;end if;

if (switch2='1'and switch20='0') then temp:=temp+10;sig\_pay<=temp;end if;

if (switch3='1'and switch30='0') then temp:=temp+20;sig\_pay<=temp;end if;

end if;

end if;

end process;

end Behavioral;

11.找零出票模块

找零出票模块中我们使用了verilog语言，只是因为我们对VHDL语言不熟悉导致不知道如何使用乘法。这里我们输入票价，票数，状态标志位，已投币数并输出总价和找零。

逻辑如下，当状态标志位置0时，即选择了终点线与终点站。我们将票价和票数相乘得到总价，再将已投币数减去总价算出找零。当标志位置1时，即直接选择票价。因为直接选择票价意味着票数为1，我们直接将已投币数减去票价，得到找零。

【代码】

module computechange(

input clk,

input flag0,

input[31:0]get\_amount,

input[31:0]ticket\_price,

input[31:0]get\_real\_pay,

input[31:0]get\_price,

input[3:0]get\_present\_state,

output[31:0]change,

output[31:0]total

);

reg [31:0] total\_price ;

reg [31:0] rest\_money ;

assign change = rest\_money;

assign total = total\_price;

always @(posedge clk )

begin

if (get\_present\_state == 4'b0000)

begin

total\_price = 0;

rest\_money = 0;

end

if (get\_present\_state == 4'b1000)

begin

total\_price = get\_price \* get\_amount;

if(flag0 == 0)

rest\_money = get\_real\_pay - total\_price;

if(flag0 == 1)

rest\_money = get\_real\_pay - ticket\_price;

end

if (get\_present\_state == 4'b1100)

begin

total\_price = get\_price \* get\_amount;

if(flag0 == 0)

rest\_money = get\_real\_pay - total\_price;

if(flag0 == 1)

rest\_money = get\_real\_pay - ticket\_price;

end

if (get\_present\_state == 4'b0111)

begin

total\_price = get\_price \* get\_amount;

if(flag0 == 0)

rest\_money = get\_real\_pay - total\_price;

if(flag0 == 1)

rest\_money = get\_real\_pay - ticket\_price;

end

end

endmodule

12.退币模块

本模块功能较为简单，实际上只是将之前投币模块的输出端口传递过来，变为此处的输出端口，但是名称变成了退币端口。经过系统的调试后，我们发现这一模块其实承担的作用并不明显，某种程度上也可省略，但是为了状态机的完整与可读性，我们仍然保留了这一模块。

【代码】

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;--使用函数conv\_std\_logic\_vector(m,n)的前提

entity refund is

Port (clk:in std\_logic;

get\_real\_pay:in std\_logic\_vector(31 downto 0);

get\_present\_state:in std\_logic\_vector(3 downto 0)

);

end refund;

architecture Behavioral of refund is

signal sig\_real\_pay:std\_logic\_vector(31 downto 0);

signal sig\_refund:std\_logic\_vector(31 downto 0);

begin

process(clk,get\_present\_state)

begin

if(clk'event and clk='1') then

if(get\_present\_state="1001") then

--显示退款即可，refund就是get\_real\_pay

sig\_real\_pay<=get\_real\_pay;

sig\_refund<=sig\_real\_pay;

end if;

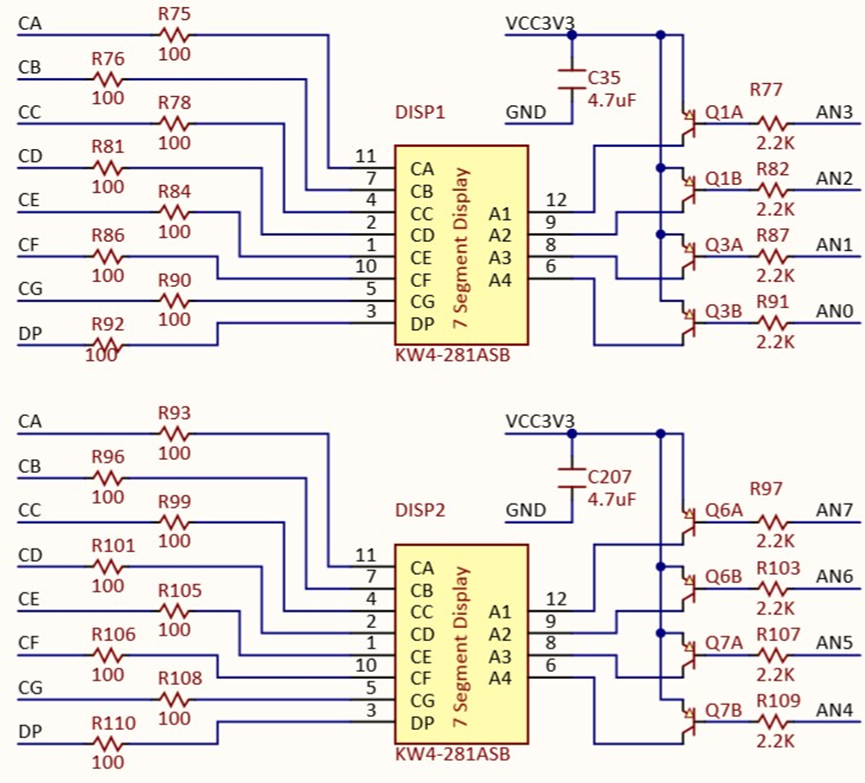
end if;

end process;

end Behavioral;

13.7段显示数码管模块

在7段显示数码管模块中，输入为时钟，32bit的8个数码管要显示的字符（即每个字符被编码成4bit，从1，2，3到E,F），8bit的seg\_able(每一位对每一个数码管是否被点亮做判断)，输出为8bit的seg和8bit的an（对应到硬件电路）



图片包含 屏幕截图

描述已自动生成

【代码】

module seven\_segment\_disp(

input clk,

input [31:0]dispdata,

input [7:0] seg\_able,

output reg [7:0]segg,

output reg [7:0]an

);

reg [31:0] divclk\_cnt = 0;

reg divclk = 0;

reg [2:0] loop\_bit = 3'b000;

reg [3:0] loop\_data ;

parameter maxcnt = 25000;

always@(posedge clk)

begin

if(divclk\_cnt == maxcnt)

begin

divclk = ~divclk;

divclk\_cnt = 0;

end

else

begin

divclk\_cnt = divclk\_cnt + 1'b1;

end

end

always@(posedge divclk)

begin

case(loop\_bit)

3'b000:

begin

if(seg\_able[0])

begin

an = 8'b11111110;

end

else

begin

an = 8'b11111111;

end

loop\_data[3:0] = dispdata [3:0];

end

3'b001:

begin

if(seg\_able[1])

begin

an = 8'b11111101;

end

else

begin

an = 8'b11111111;

end

loop\_data[3:0] = dispdata [7:4];

end

3'b010:

begin

if(seg\_able[2])

begin

an = 8'b11111011;

end

else

begin

an = 8'b11111111;

end

loop\_data[3:0] = dispdata [11:8];

end

3'b011:

begin

if(seg\_able[3])

begin

an = 8'b11110111;

end

else

begin

an = 8'b11111111;

end

loop\_data[3:0] = dispdata [15:12];

end

3'b100:

begin

if(seg\_able[4])

begin

an = 8'b11101111;

end

else

begin

an = 8'b11111111;

end

loop\_data[3:0] = dispdata [19:16];

end

3'b101:

begin

if(seg\_able[5])

begin

an = 8'b11011111;

end

else

begin

an = 8'b11111111;

end

loop\_data[3:0] = dispdata [23:20];

end

3'b110:

begin

if(seg\_able[6])

begin

an = 8'b10111111;

end

else

begin

an = 8'b11111111;

end

loop\_data[3:0] = dispdata [27:24];

end

3'b111:

begin

if(seg\_able[7])

begin

an = 8'b01111111;

end

else

begin

an = 8'b11111111;

end

loop\_data[3:0] = dispdata [31:28];

end

endcase

loop\_bit = loop\_bit + 1'b1;

end

always@(loop\_data)

begin

case(loop\_data)

4'h0:segg=8'hc0;

4'h1:segg=8'hf9;

4'h2:segg=8'ha4;

4'h3:segg=8'hb0;

4'h4:segg=8'h99;

4'h5:segg=8'h92;

4'h6:segg=8'h82;

4'h7:segg=8'hf8;

4'h8:segg=8'h80;

4'h9:segg=8'h90;

4'ha:segg=8'h88;

4'hb:segg=8'h83;

4'hc:segg=8'hc6;

4'hd:segg=8'ha1;

4'he:segg=8'h86;

4'hf:segg=8'h8e;

endcase

end

endmodule

三、仿真与测试

我们对BRAM模块做了仿真并编写了仿真文件。

【代码】

module mem\_gen;

reg clk =0;

reg rst = 0;

reg we = 0;

reg en =1;

reg[15:0]din=0;

reg[9:0]addr=0;

wire[15:0]dout;

reg[9:0]cnt1 = 0;

reg[9:0]cnt2 = 0;

VGA ss(clk,rst,en,we,din,addr,dout);

always #10 clk = ~clk;

always @(posedge clk)

begin

if(cnt1 == 8)

begin

cnt1 = 0;

cnt2 = cnt2 + 1;

end

else

begin

cnt1 = cnt1+1;

end

end

always @(negedge clk)

begin

din = cnt1;

addr = cnt1;

if(cnt2[0] == 0) we = 0;

else we = 1;

end

endmodule

在这次仿真中，BRAM配置为简单单端口，宽度为16bit，深度为我们coe文件存了800个数据，从800到1599。在时钟第一个上升沿，地址线上出现的数据为h’320,即d’800，在第二个上升沿出现的数据为h’321,即d’801,故模块工作正确。

图片包含 屏幕截图

描述已自动生成

四、合作分工

包括如何分工、如何协调进展以及成效与总结。

（1）分工

04017404王宸：

顶层模块（布置端口映射关系）

状态机

选择起始、终点站模块

选择票价模块、退币模块

投币模块、显示模块

按键消抖模块

04017537乔竞：

7段显示数码管模块

BRAM模块

编写python写入coe文件

计算票价模块

VGA显示研究

（2）协调进展

第一周，我们学习了几例官方教程，并熟悉了VIVADO中从编写程序到烧录的过程。

第二周，正式开始分析题目，厘清系统流程和结构，并简单分配了各自任务。之后，我们开始分两路开始研究与编写程序，基本完成了状态机模块和七段显示数码管模块。

第三周，我们继续按照计划完成了包括选择起始站点、选择终点站点、选择票价、按键消抖等等子模块的编写，学习与尝试了VGA的实现。这一周，我们在计算票价的问题上花了很久的时间讨论与尝试，最终确定用写入票价并读取的方式实现。

第四周，完成了票价的写入与读取，并计算总票价和找零，并完善了各个子模块与顶层模块直接的衔接关系。最后经过测试与调整，终于完成了地铁售票机的制作。

（3）成效与总结

经过本次课程，我们发现了自己在合作这一方面还存在很大的不足。尽管最后勉强顺利完成了基本要求，但是我们的进程安排不够系统和周到，在前几周完成的工作量较少，导致后期时间较为紧张，剩余测试与改进的空间缩小，因此也对许多出现的问题作了简化处理，没有深入研究下去。此外，合作应该是一个互相帮助、共同进步的过程，但我们往往分头行动，没有完整的交流与讨论，这也是工作效率不高的一个主要原因。

五、设计中遇到的问题与解决措施

（1）在找零出票环节，需要对之前的操作进行一次判断，即是直接选票价后投币，还是选终点站后选票数再投币，这两种情况将对应不同的找零代码。一开始，我们试图统一两种情况对应的找零算法，但是并不容易。后来，我们准备设置一个标志位，令其在选当前站里为0，在选票价里为1，则如果检测到flag1是1的话就按照总票价进行找零，否则按照快速选票价的单票价进行找零。但这一过程容易导致标志位的混乱，最终我们只考虑了经过快速选票价的情况，即在等待状态时标志位为0，一旦经过选票价状态，即将标志位置1，这样后续的判断就不再出现问题了。

（2）在读取BRAM的模块中。我当时将读取VGA的时钟同时给了端口A和B，导致读取内存也是用了分频后的31.5Mhz这就可能导致时序有问题。后来我们将时钟频率恢复成原来的100Mhz后，整个状态机正常了。

六、课程总结

通过这门课，我们在FPGA上完成了一个状态机，学会了Verilog和VHDL，同时也学会了写仿真文件。我们学会了管脚约束，也学会了时钟分频和BRAM的配置。最后很可惜的是虽然花了时间在VGA显示上，但我们没有成功地显示照片。总的来说这是一个很有意思的项目。