Lab Report No. 04:

Registers

By:

Jiaqing Wan(16), Jiaxin Zheng(19)

Instructor: Dr. Xi Zhou

Classnumber: ZM1501

NAU ID:

5358836, 5358840

October 22, 2017

I. INTRODUCTION

A. Objectives

At the completion of this lab, the student will be able to:
1. Complete a VHDL design from a functional specification (write VHDL, simulate functional behavior, and demonstrate functional behavior on the Cyclone V demo board) for sequential circuits.

B. Background knowledge

The Accumulator is a register that stores results from the ALU and later feeds those results back into the ALU for more processing. Its connection to the ALU is:

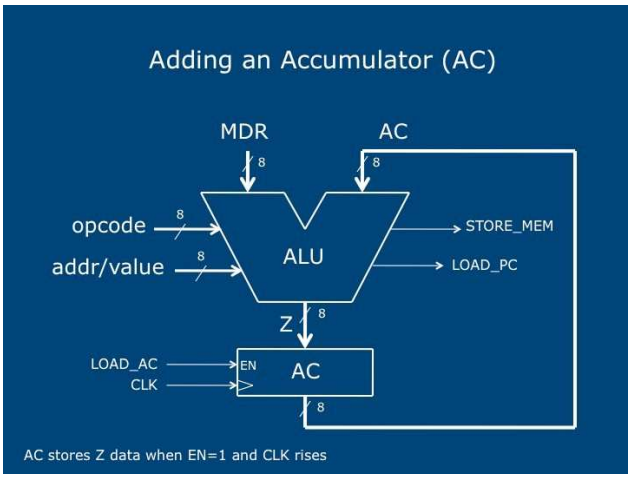

FIG. I.1 Accumulator in ALU

The Instruction Register holds the instruction that was most recently fetched from memory. As described below, the instruction field is two bytes (16 bits) wide, with the upper byte being the opcode and the lower byte being the address or value information. Since our data bus is only 8 bits wide, we will usually fetch instructions in two different memory read operations. The first read cycle will fetch the opcode information, which will be stored into the upper byte of the IR. If the opcode happens to be for an instruction that includes an

address or value, that information will be fetched from memory and stored into the lower byte of the IR.

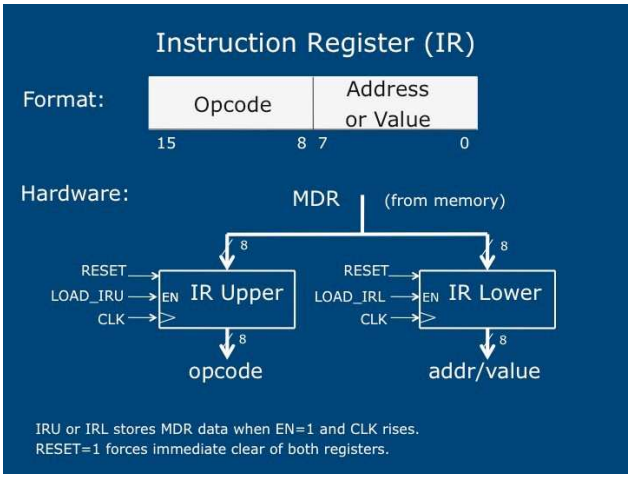

FIG. I.2 The Instruction Register format and hardware

The Program Counter is actually an 8-bit wide sequential device that can be cleared (store a zero), loaded with a new value, or incremented (add 1 to the current value). Since the PC contains the memory address of an instruction, we will use its contents to send to memory as an address.

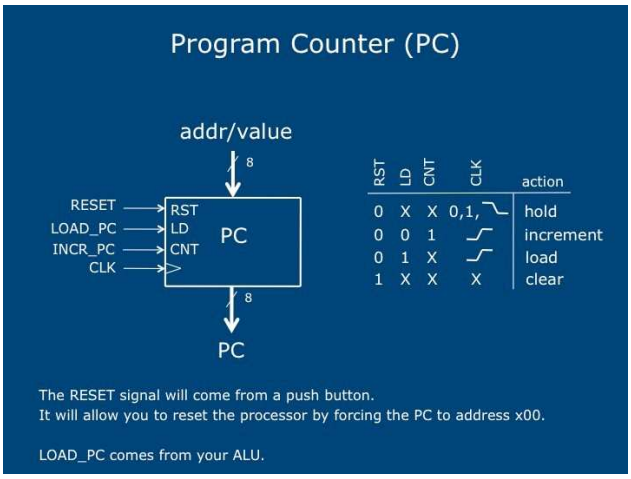

FIG. I.3 The Program Counter state machine

C. Design strategy

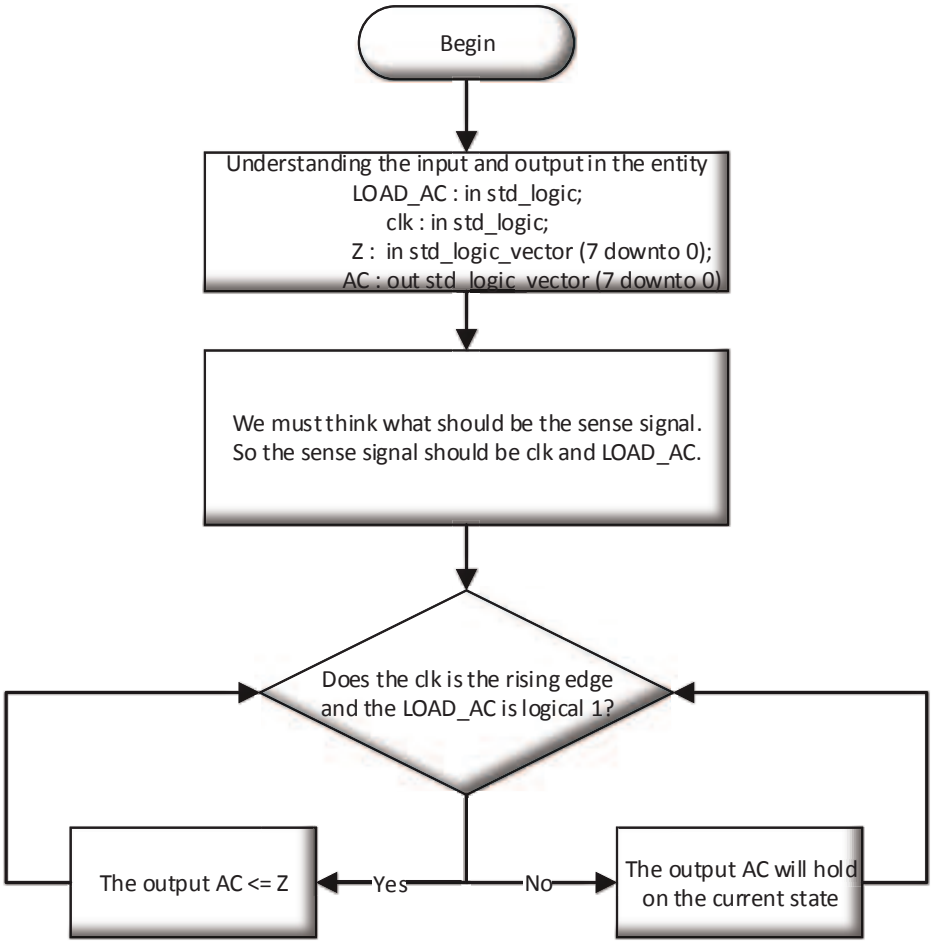

FIG. I.4 The accumulator design strategy

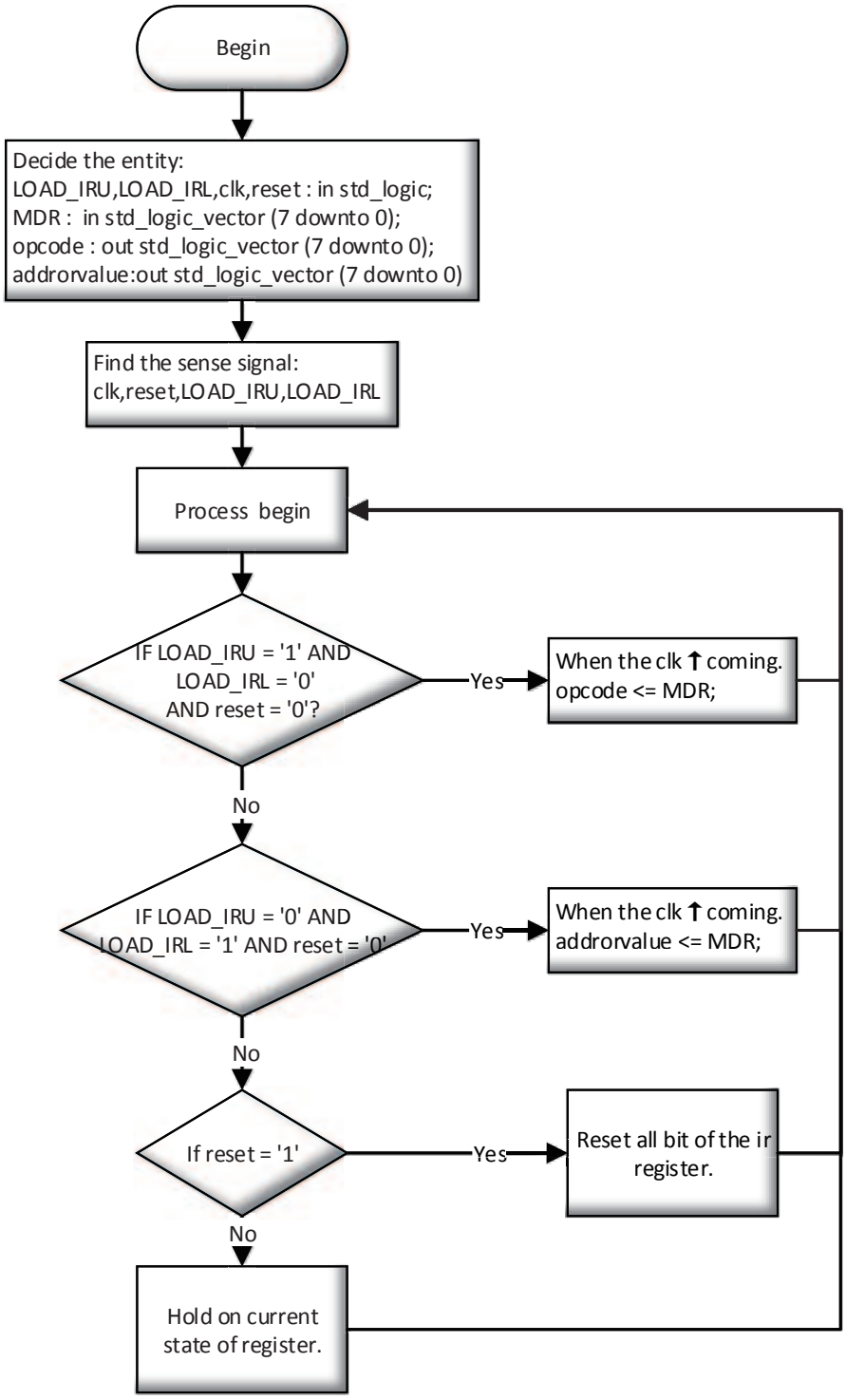

FIG. I.5 The Instruction Register design strategy

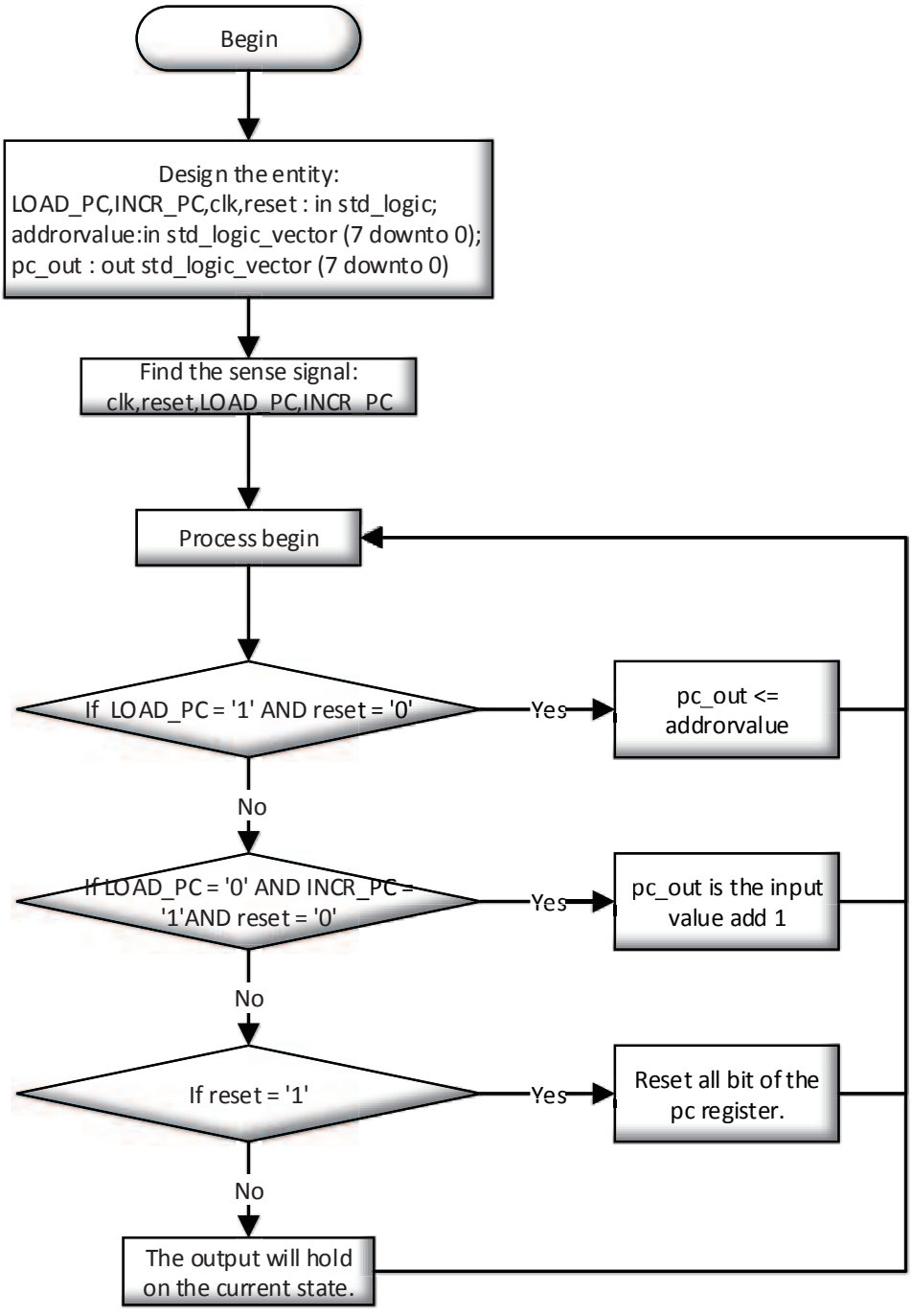

FIG. I.6 The Program Counter design strategy

D. Simulation strategy

TABLE I Acculator Demonstration

LOAD_AC	Input signal	clk	Output
low	×	↑	Hold
high	Z	↑	Z
×	×	not↑	Hold

TABLE II Instruction Register Demonstration

reset	LOAD_IRU	LOAD_IRL	MDR[7..0]	clk	seg3, seg2	seg1, seg0
up	×	×	×	×	0	0
down	high	low	Q	↑	Q	Hold
down	low	high	Q	↑	Hold	Q
Other condition: Not change						

TABLE III Program Counter Demonstration

reset	LOAD_PC	INCR_PC	addrorvalue	clk	seg1,seg0
up	×	×	×	×	0
down	high	×	Q	↑	Q
down	low	high	Q	↑	Q+1
Other condition: Not change.					

E. Demonstration strategy

TABLE IV The Accumulator Set

Input Signal			Output signal
Z[7..0]	LOAD_AC	clk	the HEX1 is upper 4 bit
SW[7..0]	SW8	KEY0	the HEX2 is lower 4 bit

TABLE V Instruction Register Set

Input signal					Output
reset	LOAD_IRU	LOAD_IRL	MDR[7..0]	clk	seg7 display the
KEY1	SW9	SW8	SW[7..0]	KEY0	number of SW[7..0]

TABLE VI Program Counter Set

Input					Output
reset	LOAD_PC	INCR_PC	addrorvalue	clk	pc_out
KEY1	SW8	SW9	SW[7..0]	KEY0	seg1,seg0

II. LAB DETAILS

A. Instructor questions in the lab instruction

There is no instructor questions in this lab.

B. VHDL code

accu.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;

entity accu is
   port(
   LOAD_AC : in std_logic;
   clk : in std_logic;
   Z :  in std_logic_vector (7 downto 0);
   AC : out std_logic_vector (7 downto 0)
   );

   end entity accu ;

      architecture Accumulator of accu is
      begin
      ----------------------------
         process(LOAD_AC, clk)
         begin
               IF clk'event AND clk = '1' AND LOAD_AC = '1' then
                  AC <= Z;--load Z
               END IF;
         end process;
         --other condition will hold on
      ----------------------------
      end architecture Accumulator ;
```

IR.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;

entity IR is
   port(
   LOAD_IRU,LOAD_IRL,clk,reset : in std_logic;
   MDR :  in std_logic_vector (7 downto 0);--input
   opcode : out std_logic_vector (7 downto 0);--output
   addrorvalue:out std_logic_vector (7 downto 0)--output
   );

   end entity IR ;

      architecture  InstructionRegister of IR is
      begin
      ------------------------------
         process(clk,reset,LOAD_IRU,LOAD_IRL)
         begin
            ------------------------------
            --case1 load opcode
            IF LOAD_IRU = '1' AND LOAD_IRL = '0' AND reset = '0'  THEN
               IF clk'event AND clk = '1' THEN
                  opcode <= MDR;
               END IF;
            END IF;
            ------------------------------
            --case2 load addr or value
            IF LOAD_IRU = '0' AND LOAD_IRL = '1' AND reset = '0' THEN
               IF clk'event AND clk = '1' THEN
                  addrorvalue <= MDR;
               END IF;
            END IF;
            ------------------------------
            --case3 reset
            IF reset = '1' then
                  opcode <= "00000000";
                  addrorvalue <= "00000000";
            END IF;
            ------------------------------
            --other condition hold on
         end process;
      ------------------------------
      end architecture  InstructionRegister ;
```

pc.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity pc is
   port(
   LOAD_PC,INCR_PC,clk,reset : in std_logic;
   addrorvalue:in std_logic_vector (7 downto 0);
   pc_out : out std_logic_vector (7 downto 0)
   );
```

```
end entity pc ;

   architecture  ProgramCounter of pc is
      signal S : std_logic_vector(7 downto 0);
   begin
      ------------------------------
      process(clk,reset,LOAD_PC,INCR_PC)
      begin
         --case 1 :load the data
         ------------------------------------------------------------------------
         IF  (LOAD_PC = '1' AND reset = '0')AND (clk'event AND clk = '1')THEN
            pc_out <= addrorvalue;
            S <= addrorvalue;
         END IF;
         --case 2: count the address
         ------------------------------------------------------------------------
         IF (LOAD_PC = '0' AND INCR_PC = '1'AND reset = '0') AND (clk'event AND clk = '1')THEN
            S <= S + 1; --because the data is paralle ;next rise edge will change
            pc_out <= conv_std_logic_vector (unsigned(S)+1, 8);--so we must add at there
         END IF;
         --case 3: reset
         ------------------------------------------------------------------------
         IF reset = '1' then
            pc_out <= (others => '0');
         END IF;
         ------------------------------------------------------------------------
      end process;
      ------------------------------
   end architecture  ProgramCounter ;
```

C. Block diagram

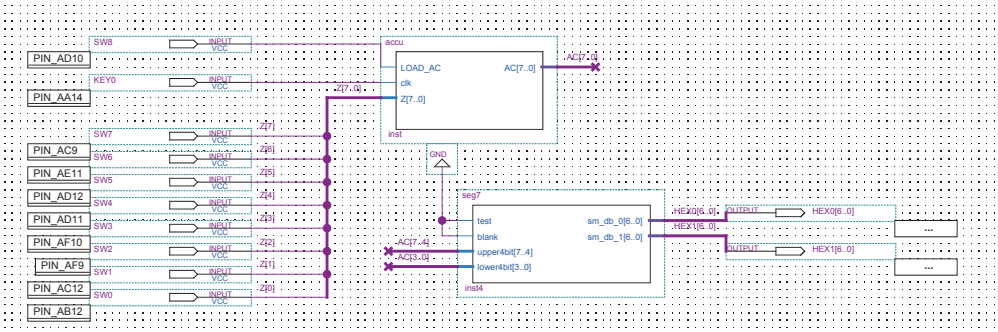

FIG. II.1 accu_tb Block diagram

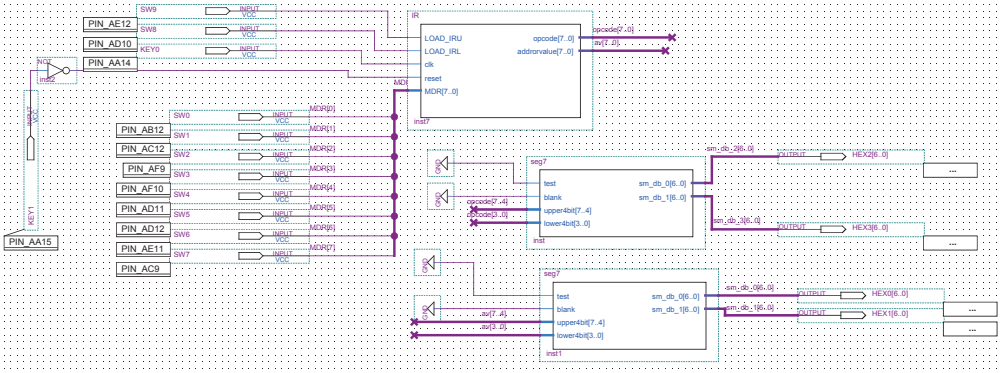

FIG. II.2 IR_tb Block diagram

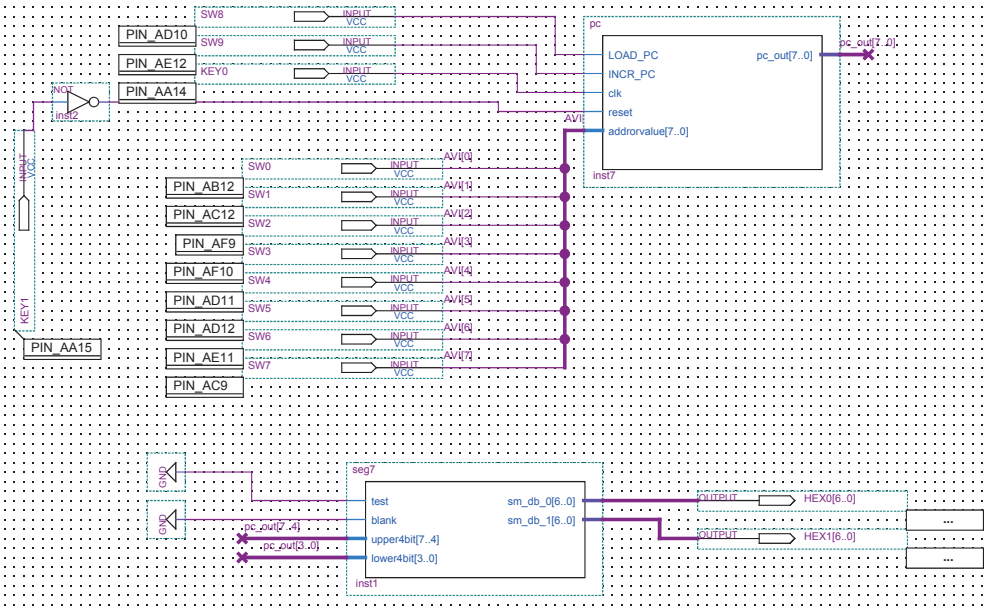

FIG. II.3 pc_tb Block diagram

D. ModelSim force file

accu_tb.txt:

```
force clk 0 0ns, 1 50ns -r 100ns
force LOAD_AC 1 0ns, 0 800ns -r 1600ns

#when LOAD_AC is 1,output will follow.
force Z 00000001
```

```
run 100ns
force Z 00000010
run 100ns
force Z 00000100
run 100ns
force Z 00001000
run 100ns
force Z 00010000
run 100ns
force Z 00100000
run 100ns
force Z 01000000
run 100ns
force Z 10000000
run 100ns

#when LOAD_AC is 0, output will hold
force Z 00000001
run 100ns
force Z 00000010
run 100ns
force Z 00000100
run 100ns
force Z 00001000
run 100ns
force Z 00010000
run 100ns
force Z 00100000
run 100ns
force Z 01000000
run 100ns
force Z 10000000
run 100ns
```

IR_tb.txt:

```
force reset 1
run 100ns

#test load opcode
force reset 0

force clk 0 0ns, 1 50ns -r 100ns
force MDR 11111111
force LOAD_IRU 1
force LOAD_IRL 0
run 200ns
force MDR 00000000
run 200ns

# test load addr or value
force MDR 11111111
force LOAD_IRL 1
force LOAD_IRU 0
run 200ns
force MDR 00000000
run 200ns
```

pc_tb.txt:

```
force clk 0 0ns,1 50ns -r 100ns;#clk set

#test for reset function,all case are 0;
force reset 1
force addrorvalue 11111111
force LOAD_PC 1
force INCR_PC 0
run 100 ns
force LOAD_PC 0
force INCR_PC 1
run 100 ns
force LOAD_PC 0
force INCR_PC 0
run 100 ns
force LOAD_PC 1
force INCR_PC 1
run 100 ns

#test for load
force reset 0
force addrorvalue 00000000
force LOAD_PC 1
run 200ns;#follow addrorvalue when clk at rising edge
force addrorvalue 11111111
run 100ns;#follow addrorvalue when clk at rising edge

#test for increment
force LOAD_PC 0
force INCR_PC 1
run 400ns; #add 1; shown 0 at begin;
force LOAD_PC 1
force INCR_PC 0
force addrorvalue 00000000
run 100ns; #load value 00000000
force INCR_PC 1
force LOAD_PC 0
run 400ns;# add 1; shown 1 at begin
```

E. Simulation results and detailed explanation

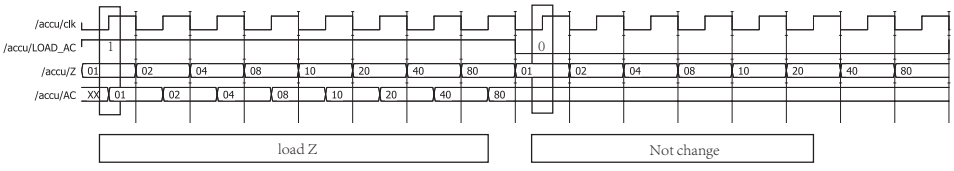

FIG. II.4 accu_tb Wave

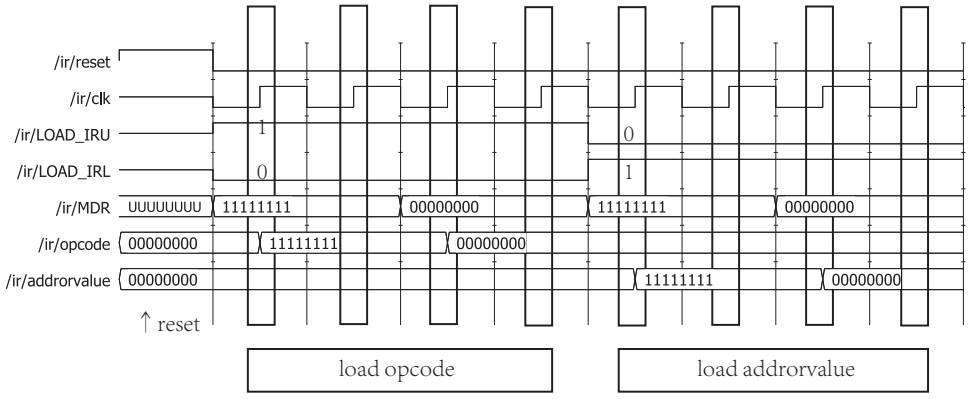

FIG. II.5 IR_tb Wave

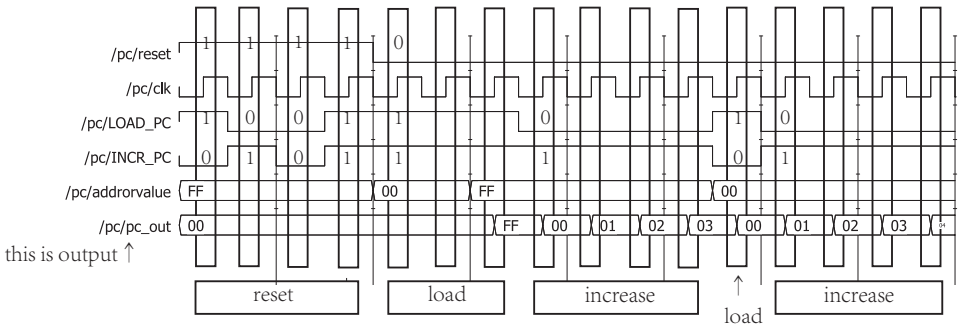

FIG. II.6 pc_tb Wave

F. Board demonstration result

TABLE VII Acculator Demonstration

LOAD_AC	Input signal	clk	Output
low	×	↑	Hold
high	Z	↑	Z
×	×	not↑	Hold

TABLE VIII Instruction Register Demonstration

reset	LOAD_IRU	LOAD_IRL	MDR[7..0]	clk	seg3, seg2	seg1, seg0
up	×	×	×	×	0	0
down	high	low	Q	↑	Q	Hold
down	low	high	Q	↑	Hold	Q
Other condition: Not change						

TABLE IX Program Counter Demonstration

reset	LOAD_PC	INCR_PC	addrorvalue	clk	seg1,seg0
up	×	×	×	×	0
down	high	×	Q	↑	Q
down	low	high	Q	↑	Q+1
Other condition: Not change.					

accu_tb.sof:

The logical is the same as Table in the Board demonstration.

IR_tb.sof:

The logical is the same as Table in the Board demonstration.

pc_tb.sof:

The logical is the same as Table in the Board demonstration.

III. CONCLUSION

The lab is successful.

I realize all function we should achieve. The most difficulty things is the Instructor questions.

I refer to manual and ask classmates that I solved it.