FPGA系統設計\_第十三章\_步進馬達專題

**10867024 廖育賢**

# **objective- THE PROBLEM AND PURPOSE**

步進馬達專題的重點在於整體的控制技術，並配合LED及鍵盤來設定命令與參數，建立步進馬達速度控制、定位控制與加減速控制實驗。

# **procedure – DESIGN methods**

## 外部電路配置:

1. LED鍵盤模組 : 顯示與設定
2. 步進馬達模組 : 四線圈與驅動器。

## 內部電路設計:

1. 命令暫存器 : REG0下達移動距離命令，REG1以BITO控制轉動方向DIR訊號，REG2家減速的最高速限。
2. 剩餘距離計數 : 以下數計數器計算。
3. 加速距離計數 : 以上數計數器計算。
4. 位置控制電路 : 依據目前位置與速度，更新目前的控制狀態STS。
5. 加減速控制 : 依據目前加減速狀態STS，更新SPD速度命令。
6. 積分除頻電路 : 將速度命令SPD轉成指定頻率PLS脈衝訊號。
7. 正反轉控制 : 由PLS脈衝訊號與DIR轉動方向，產生正反轉的計數SEQ訊號。
8. 半步查表處理 : 由0~7的SEQ訊號查出對應的半步控制表格，並送至步進馬達的四個線圈。

# **simulation results**

## program codes

### **ch013.vhd**

--The IEEE standard 1164 package, declares std\_logic, rising\_edge(), etc.

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.std\_logic\_arith.all;

use IEEE.std\_logic\_unsigned.all;

--This package is a version of the Synopsys package and has been

-- optimized for use with the Express compiler.

--library SYNOPSYS;

--use SYNOPSYS.attributes.all;

--

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- CH13C : stepping motor control (test #C)

-- designed by Pei-Chong Tang, Feb. 1999

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

--

entity CH13C is

port

( CLK : in STD\_LOGIC; --4MHz clock

XA : out STD\_LOGIC\_VECTOR (3 downto 0); --LED scanning

XB : out STD\_LOGIC\_VECTOR (7 downto 0); --LED segment

XC : in STD\_LOGIC\_VECTOR (3 downto 0); --keyboard input

YA : out STD\_LOGIC\_VECTOR (3 downto 0) --stepping output

);

end CH13C;

architecture CH13C\_ARCH of CH13C is

component MONITOR

port

( CLK : in STD\_LOGIC; --system clock

DB : inout STD\_LOGIC\_VECTOR (7 downto 0); --data bus

A : out STD\_LOGIC\_VECTOR (3 downto 0); --address bus

RD : out STD\_LOGIC; --read control

SWR : out STD\_LOGIC; --write control

XA : out STD\_LOGIC\_VECTOR (3 downto 0); --LED scanning

XB : out STD\_LOGIC\_VECTOR (7 downto 0); --LED segment

XC : in STD\_LOGIC\_VECTOR (3 downto 0); --keyboard input

FREQ: out STD\_LOGIC --test frequency

);

end component;

signal DB : STD\_LOGIC\_VECTOR (7 downto 0);

signal A : STD\_LOGIC\_VECTOR (3 downto 0);

signal RD,SWR : STD\_LOGIC;

signal STS : STD\_LOGIC\_VECTOR (1 downto 0);

signal SEQ : STD\_LOGIC\_VECTOR (2 downto 0);

signal STEP : STD\_LOGIC\_VECTOR (3 downto 0);

signal SPD : STD\_LOGIC\_VECTOR (7 downto 0);

signal LMT : STD\_LOGIC\_VECTOR (7 downto 0);

signal LENGTH : STD\_LOGIC\_VECTOR (7 downto 0);

signal ACCLNG : STD\_LOGIC\_VECTOR (7 downto 0);

signal DIR,PLS,FREQ: STD\_LOGIC;

signal CS0 : STD\_LOGIC;

begin

--

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

--

-- system connection for the experiment

--

SYSTEM\_CONNECT : block

begin

U1: MONITOR port map (CLK, DB, A, RD, SWR, XA, XB, XC, FREQ);

YA <= STEP;

end block SYSTEM\_CONNECT;

--

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

--

-- command register

--

COMMAND\_REGISTER : block

signal CS,CS1,CS2 : STD\_LOGIC;

begin

process (CLK)

begin

if CLK'event and CLK='1' then

if CS1='1' then

DIR <= not DIR; --DIR: register #1

end if;

if CS2='1' then

LMT <= DB; --LMT: register #2

end if;

end if;

end process;

CS <= SWR and not A(3) and not A(2);

CS0 <= CS and not A(1) and not A(0);

CS1 <= CS and not A(1) and A(0);

CS2 <= CS and A(1) and not A(0);

end block COMMAND\_REGISTER;

--

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

--

-- position control

--

POSITION\_CONTROL : block

signal EC,TRG0,TRG1,TRG2,TRG3 : STD\_LOGIC;

begin

process (CLK)

begin

if CLK'event and CLK='1' then

if EC='1' then

STS <= STS + 1;

end if;

end if;

end process;

EC <= TRG0 or TRG1 or TRG2 or TRG3;

TRG0 <= '1' when STS=0 and CS0='1' else '0';

TRG1 <= '1' when STS=1 and (LENGTH<=ACCLNG or SPD>=LMT) else '0';

TRG2 <= '1' when STS=2 and LENGTH<=ACCLNG else '0';

TRG3 <= '1' when STS=3 and LENGTH=0 else '0';

end block POSITION\_CONTROL;

--

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

--

-- position counting

--

POSITION\_COUNTING : block

signal EC : STD\_LOGIC;

begin

process (CLK)

begin

if CLK'event and CLK='1' then

if EC='1' then

if CS0='1' then

LENGTH <= DB;

else

LENGTH <= LENGTH - 1;

end if;

end if;

end if;

end process;

EC <= CS0 or PLS;

end block POSITION\_COUNTING;

--

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

--

-- accelerate counting

--

ACC\_COUNTING : block

signal EC,RST : STD\_LOGIC;

begin

process (CLK,RST)

begin

if RST='1' then

ACCLNG <= "00000000";

elsif CLK'event and CLK='1' then

if EC='1' then

ACCLNG <= ACCLNG + 1;

end if;

end if;

end process;

RST <= '1' when STS=0 else '0';

EC <= '1' when STS=1 and PLS='1' else '0';

end block ACC\_COUNTING;

--

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

--

-- accelerate control

--

ACC\_CONTROL : block

signal EC,DIR,SET : STD\_LOGIC;

begin

process (CLK)

begin

if SET='1' then

SPD <= "00000001";

elsif CLK'event and CLK='1' then

if EC='1' then

if DIR='1' then

SPD <= SPD - 1;

--SPD <= SPD;

else

SPD <= SPD + 1;

--SPD <= SPD;

end if;

end if;

end if;

end process;

SET <= '1' when STS=0 or SPD=0 else '0';

EC <= FREQ and STS(0);

DIR <= '0' when STS=1 else '1';

end block ACC\_CONTROL;

--

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

--

-- speed control

--

SPEED\_CONTROL : block

signal BUF : STD\_LOGIC\_VECTOR (8 downto 0);

signal EC,DLY : STD\_LOGIC;

begin

process (CLK)

begin

if CLK'event and CLK='1' then

DLY <= BUF(8);

if EC='1' then

BUF <= BUF + ('0' & SPD);

end if;

end if;

end process;

EC <= FREQ;

PLS <= BUF(8) xor DLY when STS/=0 else '0';

end block SPEED\_CONTROL;

--

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

--

-- test step-table for stepping motor control

--

STEP\_MOVING : block

signal EC : STD\_LOGIC;

begin

process (CLK)

begin

if CLK'event and CLK='1' then

if EC='1' then

if DIR='1' then

SEQ <= SEQ - 1;

else

SEQ <= SEQ + 1;

end if;

end if;

end if;

end process;

EC <= PLS;

end block STEP\_MOVING;

--

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

--

-- table mapping for stepping motor

--

TABLE\_MAPPING : block

begin

STEP <= "0001" when SEQ=0 else

"0011" when SEQ=1 else

"0010" when SEQ=2 else

"0110" when SEQ=3 else

"0100" when SEQ=4 else

"1100" when SEQ=5 else

"1000" when SEQ=6 else

"1001";

end block TABLE\_MAPPING;

--

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- end of architechture

--

end CH13C\_ARCH;

--The IEEE standard 1164 package, declares std\_logic, rising\_edge(), etc.

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.std\_logic\_arith.all;

use IEEE.std\_logic\_unsigned.all;

--This package is a version of the Synopsys package and has been

-- optimized for use with the Express compiler.

--library SYNOPSYS;

--use SYNOPSYS.attributes.all;

--

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- MONITOR : system monitor with LED and keyboard

-- designed by Pei-Chong Tang, Feb. 1999

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

--

entity MONITOR is

port

( CLK : in STD\_LOGIC; --system clock

DB : inout STD\_LOGIC\_VECTOR (7 downto 0); --data bus

A : out STD\_LOGIC\_VECTOR (3 downto 0); --address bus

RD : out STD\_LOGIC; --read control

SWR : out STD\_LOGIC; --write control

XA : out STD\_LOGIC\_VECTOR (3 downto 0); --LED scanning

XB : out STD\_LOGIC\_VECTOR (7 downto 0); --LED segment

XC : in STD\_LOGIC\_VECTOR (3 downto 0); --keyboard input

FREQ: out STD\_LOGIC --test frequency

);

end MONITOR;

architecture MONITOR\_ARCH of MONITOR is

component KEYBOARD

port

( CLK : in STD\_LOGIC; --system clock

SAMPLE : in STD\_LOGIC; --sample point

SEQ : in STD\_LOGIC\_VECTOR(1 downto 0); --scanning sequence

INP : in STD\_LOGIC\_VECTOR(3 downto 0); --keyboard input

CODE : out STD\_LOGIC\_VECTOR(3 downto 0); --keyboard output

KIN : out STD\_LOGIC --keyboard strike

);

end component;

signal REG : STD\_LOGIC\_VECTOR (12 downto 0);

signal CODE : STD\_LOGIC\_VECTOR (3 downto 0);

signal SEQ : STD\_LOGIC\_VECTOR (1 downto 0);

signal SEG : STD\_LOGIC\_VECTOR (6 downto 0);

signal KIN,SAMPLE,BLINK : STD\_LOGIC;

begin

--

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- system connection for LED & keyboard interface

--

SYSTEM\_CONNECT : block

signal INP : STD\_LOGIC\_VECTOR (3 downto 0);

signal ENB : STD\_LOGIC\_VECTOR (3 downto 0);

begin

U1: KEYBOARD port map (CLK, SAMPLE, SEQ, INP, CODE, KIN);

DB <= REG(7 downto 0) when REG(12)='1' else

"ZZZZZZZZ";

INP <= not XC;

XA <= not ENB;

XB(7) <= (SEQ(1) and SEQ(0) and not REG(12) and BLINK) or

(SEQ(1) and not SEQ(0));

XB(6 downto 0) <= SEG when SEQ(1 downto 0)/=3 else "0000000";

ENB <= "0001" when SEQ=0 else

"0010" when SEQ=1 else

"0100" when SEQ=2 else

"1000" when SEQ=3 else

"0000";

RD <= REG(12);

SWR <= REG(12) and KIN and

not CODE(3) and CODE(2) and not CODE(1) and not CODE(0);

A <= REG(11 downto 8);

end block SYSTEM\_CONNECT;

--

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- 22-bit free counter

--

FREE\_COUNTER : block

signal Q : STD\_LOGIC\_VECTOR (21 downto 0);

signal DLY1,DLY2 : STD\_LOGIC;

begin

process (CLK)

begin

if CLK'event and CLK='1' then

DLY1 <= Q(15);

DLY2 <= Q(13);

Q <= Q+1;

end if;

end process;

BLINK <= Q(21);

SAMPLE <= not Q(15) and DLY1;

FREQ <= not Q(13) and DLY2;

SEQ <= Q(17 downto 16);

end block FREE\_COUNTER;

DIGIT0\_MONITOR : block --\*\*\*\*\* digit #0 \*\*\*\*\*

signal EC,LD,DIR : STD\_LOGIC;

begin

process (CLK)

begin

if CLK'event and CLK='1' then --CLK rising

if EC='1' then --CLK enable

if LD='1' then

REG(3 downto 0) <= DB (3 downto 0); --load register

elsif DIR='0' then

REG(3 downto 0) <= REG(3 downto 0) + 1; --up counter

else

REG(3 downto 0) <= REG(3 downto 0) - 1; --down counter

end if;

end if;

end if;

end process;

EC <= (SAMPLE and not REG(12)) or

(KIN and not CODE(3) and CODE(1) and CODE(0));

LD <= SAMPLE and not REG(12);

DIR <= CODE(2);

end block DIGIT0\_MONITOR;

DIGIT1\_MONITOR : block --\*\*\*\*\* digit #1 \*\*\*\*\*

signal EC,LD,DIR : STD\_LOGIC;

begin

process (CLK)

begin

if CLK'event and CLK='1' then --CLK rising

if EC='1' then --CLK enable

if LD='1' then

REG(7 downto 4) <= DB (7 downto 4); --load register

elsif DIR='0' then

REG(7 downto 4) <= REG(7 downto 4) + 1; --up counter

else

REG(7 downto 4) <= REG(7 downto 4) - 1; --down counter

end if;

end if;

end if;

end process;

EC <= (SAMPLE and not REG(12)) or

(KIN and not CODE(3) and CODE(1) and not CODE(0));

LD <= SAMPLE and not REG(12);

DIR <= CODE(2);

end block DIGIT1\_MONITOR;

DIGIT2\_MONITOR : block --\*\*\*\*\* digit #2 \*\*\*\*\*

signal EC,DIR : STD\_LOGIC;

begin

process (CLK)

begin

if CLK'event and CLK='1' then --CLK rising

if EC='1' then --CLK enable

if DIR='0' then

REG(11 downto 8) <= REG(11 downto 8) + 1; --up counter

else

REG(11 downto 8) <= REG(11 downto 8) - 1; --down counter

end if;

end if;

end if;

end process;

EC <= KIN and not CODE(3) and not CODE(1) and CODE(0);

DIR <= CODE(2);

end block DIGIT2\_MONITOR;

DIGIT3\_MONITOR : block --\*\*\*\*\* digit #3 \*\*\*\*\*

signal EC : STD\_LOGIC;

begin

process (CLK)

begin

if CLK'event and CLK='1' then --CLK rising

if EC='1' then --CLK enable

REG(12) <= not REG(12); --T-type F/F

end if;

end if;

end process;

EC <= KIN and not CODE(3) and not CODE(2) and not CODE(1) and not CODE(0);

end block DIGIT3\_MONITOR;

--

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- seven-segment LED transformation

--

SEVEN\_SEGMENT : block

signal D : STD\_LOGIC\_VECTOR (3 downto 0);

begin

D <= REG(11 downto 8) when SEQ=2 else "ZZZZ";

D <= REG(7 downto 4) when SEQ=1 else "ZZZZ";

D <= REG(3 downto 0) when SEQ=0 else "ZZZZ";

--GFEDCBA

SEG <= "0111111" when D=0 else

"0000110" when D=1 else

"1011011" when D=2 else

"1001111" when D=3 else

"1100110" when D=4 else

"1101101" when D=5 else

"1111101" when D=6 else

"0000111" when D=7 else

"1111111" when D=8 else

"1101111" when D=9 else

"1110111" when D=10 else

"1111100" when D=11 else

"0111001" when D=12 else

"1011110" when D=13 else

"1111001" when D=14 else

"1110001" when D=15 else

"0000000";

end block SEVEN\_SEGMENT;

--

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- end of architechture

--

end MONITOR\_ARCH;

--The IEEE standard 1164 package, declares std\_logic, rising\_edge(), etc.

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.std\_logic\_arith.all;

use IEEE.std\_logic\_unsigned.all;

--

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- KEYBOARD : one-bit keyboard debouncing

-- designed by Pei-Chong Tang, Feb. 1999

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

--

entity KEYBOARD is

port

( CLK : in STD\_LOGIC; --system clock

SAMPLE : in STD\_LOGIC; --sample point

SEQ : in STD\_LOGIC\_VECTOR(1 downto 0); --scanning sequence

INP : in STD\_LOGIC\_VECTOR(3 downto 0); --keyboard input

CODE : out STD\_LOGIC\_VECTOR(3 downto 0); --keyboard output

KIN : out STD\_LOGIC --keyboard strike

);

end KEYBOARD;

architecture KEYBOARD\_ARCH of KEYBOARD is

signal KBUF : STD\_LOGIC\_VECTOR (4 downto 0);

signal PLS1,PLS2,PLS3 : STD\_LOGIC;

signal KEY,FLT : STD\_LOGIC;

begin

--

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

--

-- keyboard sequence control

--

KEYBOARD\_SEQUENCE : block

begin

process (CLK)

begin

if CLK'event and CLK='1' then --CLK rising

PLS3<=PLS2; PLS2<=PLS1; PLS1<=SAMPLE; --delay

end if;

end process;

end block KEYBOARD\_SEQUENCE;

--

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

--

-- keyboard decoding

--

KEYBOARD\_DECODING : block

signal D : STD\_LOGIC\_VECTOR (2 downto 0);

signal EC,RST : STD\_LOGIC;

begin

process (CLK,RST)

begin

if RST='1' then

KBUF <= "00000";

elsif CLK'event and CLK='1' then

if EC='1' then

KBUF <= D & (not SEQ);

end if;

end if;

end process;

D(2 downto 0) <= "100" when INP="0001" else

"101" when INP="0010" else

"110" when INP="0100" else

"111" when INP="1000" else

"000";

EC <= PLS1 and D(2);

RST <= PLS3 and not SEQ(1) and not SEQ(0);

end block KEYBOARD\_DECODING;

--

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

--

-- keyboard latch

--

KEYBOARD\_LATCH : block

signal EC : STD\_LOGIC;

begin

process (CLK)

begin

if CLK'event and CLK='1' then --CLK rising

if EC='1' then --CLK enable

CODE <= KBUF(3 downto 0); --latch code

KEY <= KBUF(4);

end if;

end if;

end process;

EC <= PLS2 and not SEQ(1) and not SEQ(0);

end block KEYBOARD\_LATCH;

--

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

--

-- keyboard debouncing

--

KEYBOARD\_DEBOUNCING : block

signal D0,D1,EC : STD\_LOGIC;

begin

process (CLK)

begin

if CLK'event and CLK='1' then --CLK rising

if EC='1' then --CLK enable

D1<=D0; D0<=KEY; --delay

FLT <= ((D0 and D1) or FLT) and (D0 or D1); --RS-F/F

end if;

end if;

end process;

EC <= PLS3 and not SEQ(1) and not SEQ(0);

end block KEYBOARD\_DEBOUNCING;

--

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

--

-- differential signal

--

KEYBOARD\_DIFF : block

signal D0,D1 : STD\_LOGIC;

begin

process (CLK)

begin

if CLK'event and CLK='1' then --CLK rising

D1<=D0; D0<=FLT; --delay

end if;

end process;

KIN <= D0 and not D1; --differential

end block KEYBOARD\_DIFF;

--

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- end of architechture

--

end KEYBOARD\_ARCH;

### **CH013.UCF**

##############################################

# I/O mapping for USB 3s1000 FPGA board #

##############################################

NET CLK LOC = B12;

NET XA<0> LOC = N2;

NET XA<1> LOC = T1;

NET XA<2> LOC = T2;

NET XA<3> LOC = U2;

NET XB<0> LOC = M3;

NET XB<1> LOC = M4;

NET XB<2> LOC = W3;

NET XB<3> LOC = W4;

NET XB<4> LOC = Y3;

NET XB<5> LOC = Y4;

NET XB<6> LOC = W6;

NET XB<7> LOC = Y6;

NET XC<0> LOC = V1;

NET XC<1> LOC = V2;

NET XC<2> LOC = W1;

NET XC<3> LOC = W2;

NET YA<0> LOC = W10;

NET YA<1> LOC = Y10;

NET YA<2> LOC = W9;

NET YA<3> LOC = W8;

## observations

### 1.位置控制:

### 

### 速度控制(255):

