2008 年全国大学生电子设计竞赛

——信息安全技术专题邀请赛参赛作品附录

基于可重构的可信 SOPC 平台的 WSN 安全系统

WSN secure system based on
reconfigurable trusted SOPC platform

（附录）
Appendix

参赛学校：西安电子科技大学

参赛队员：崔林涛，侯方，张朕源

指导老师：张卫东

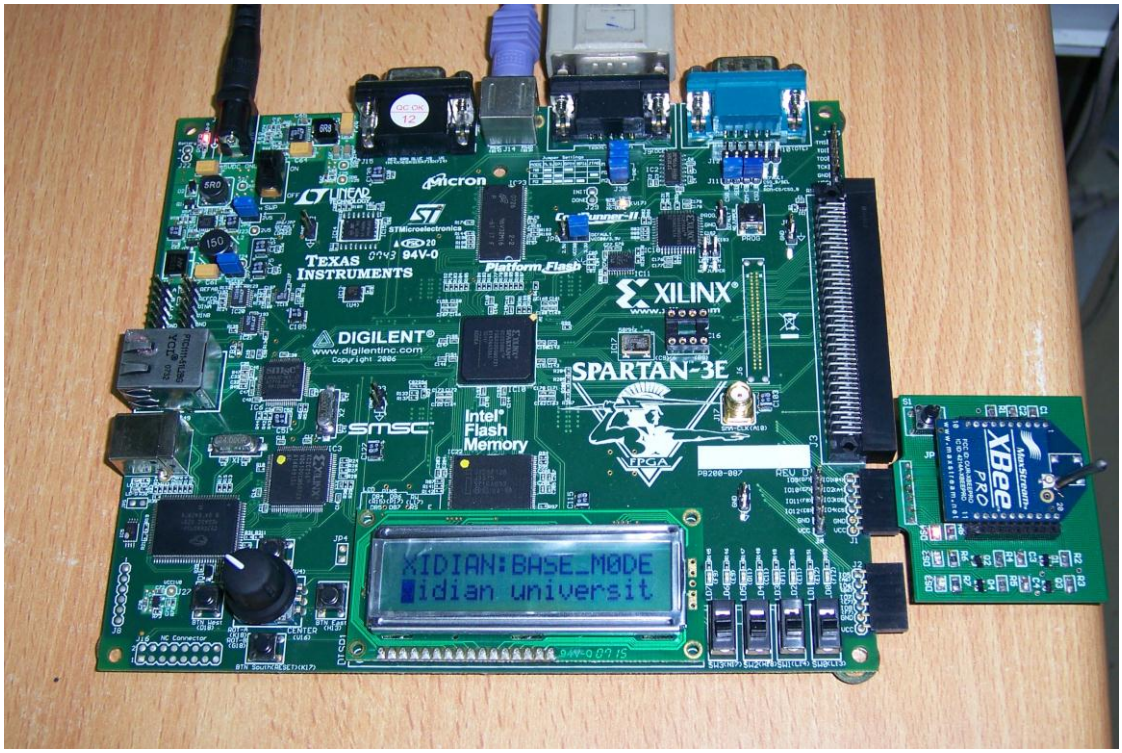
2008 年 8 月

目 录

附录 A： 系统实物图

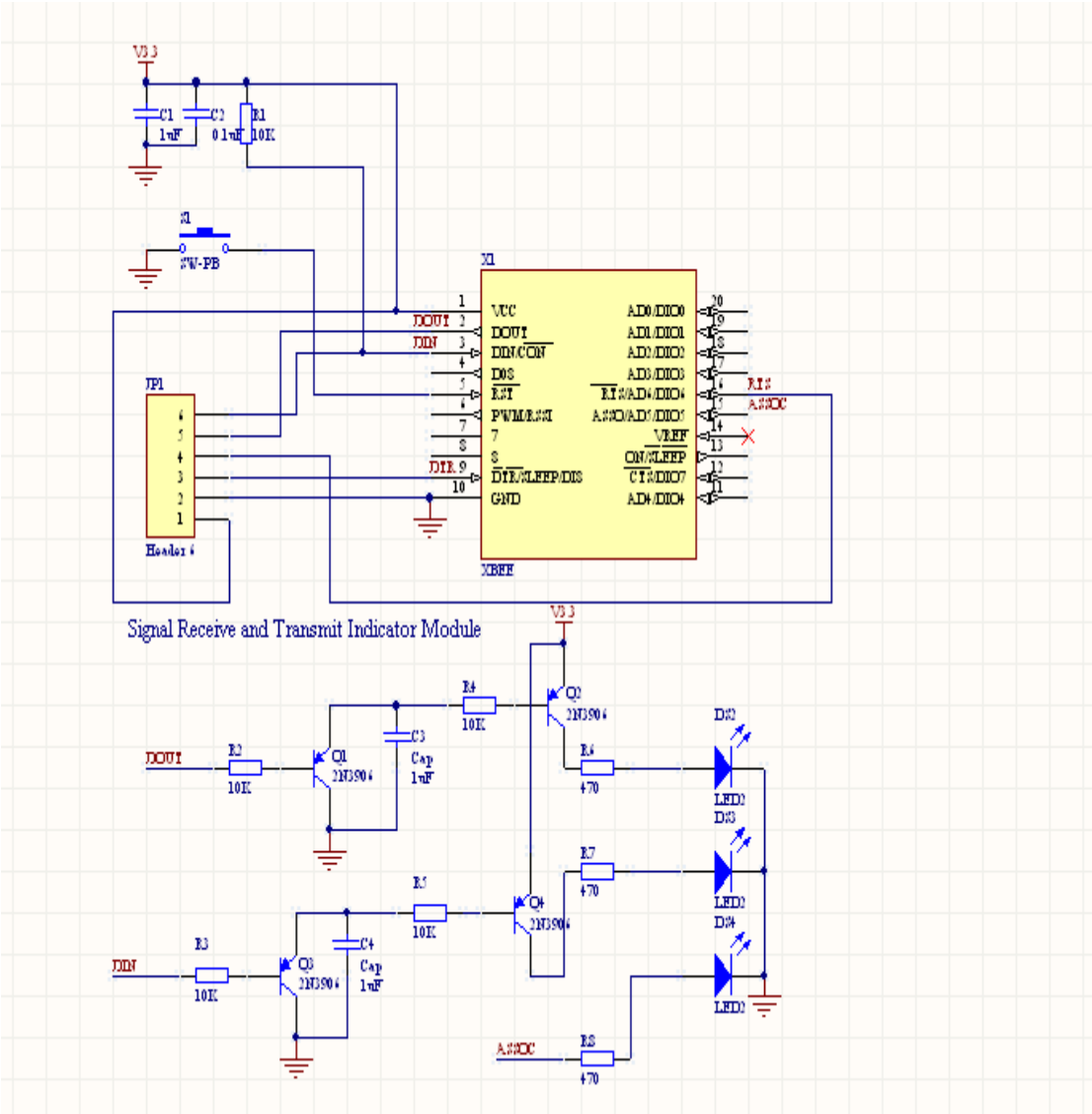
1. 系统实物图

2. 单个节点实物图


XILINX
SPARTAN-3E
DIGILENT
XIDIAN:BASE_MODE
idian universit
XBee
PRO

附录 B： 外扩硬件原理图

1. XBEE 通信模块硬件原理图



附录 C：程序清单及部分源代码

1. VHDL 源代码

顶层 VHDL 代码：top.vhd

```
----------------------------------------------------------------------------------
-- Company: xidian university
-- Engineer: cui ,zzy ,hou
-- Module Name:        top - Behavioral
----------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
library UNISIM;
use UNISIM.VComponents.all;
entity top is
   port(
            --global signals
            clk_in : IN std_logic;
            reset : IN std_logic;
         --
            --LCD_PS2 ext_ports
            rotary_a : IN std_logic;
            rotary_b : IN std_logic;
            clear_display : IN std_logic;

            ps2_clk : INOUT std_logic;
            ps2_data : INOUT std_logic;

            lcd_d : INOUT std_logic_vector(7 downto 4);
            lcd_rs : OUT std_logic;
            lcd_rw : OUT std_logic;
            lcd_e : OUT std_logic;
            strataflash_oe : OUT std_logic;
            strataflash_ce : OUT std_logic;
            strataflash_we : OUT std_logic;
         --
          --RF_top    ext_ports
            RS232_rx : IN std_logic;
```

```
            RS232_tx : OUT std_logic;

            RF_serial_rx : IN std_logic;
            RF_serial_tx : OUT std_logic;

----check port
          TPM_ready_check: out std_logic;
            data32_present:    out std_logic;
            check_rx_over:      out std_logic;
            Tpm_En_check: out std_logic;
----check port
          --Shared ext_ports
            frame_leds : OUT std_logic_vector(7 downto 0);

            TPM_clear: in std_logic
          );
end top;

architecture Behavioral of top is

      COMPONENT DCM_10
      PORT(
            CLKIN_IN : IN std_logic;
            RST_IN : IN std_logic;
            CLK : OUT std_logic;
            CLKIN_IBUFG_OUT : OUT std_logic;
            CLK0_OUT : OUT std_logic;
            LOCKED_OUT : OUT std_logic
            );
      END COMPONENT;
      signal clk : std_logic;
      signal rst: std_logic;
      signal locked: std_logic;

--cui component
      COMPONENT top_peripheral
      PORT(
            clk : IN std_logic;
            rst : IN std_logic;

            --for check
            check_rx_over: out std_logic;

            rotary_a : IN std_logic;
```

```
        rotary_b : IN std_logic;
        clear_display : IN std_logic;

        RS232_rx : IN std_logic;
        RF_serial_rx : IN std_logic;
        RS232_tx : OUT std_logic;
        RF_serial_tx : OUT std_logic;

        ps2_clk : INOUT std_logic;
        ps2_data : INOUT std_logic;
        lcd_d : INOUT std_logic_vector(7 downto 4);
        lcd_rs : OUT std_logic;
        lcd_rw : OUT std_logic;
        lcd_e : OUT std_logic;
        strataflash_oe : OUT std_logic;
        strataflash_ce : OUT std_logic;
        strataflash_we : OUT std_logic;

        frame_leds : OUT std_logic_vector(7 downto 0);


        TX_ready : OUT std_logic;

        ext_OK : IN std_logic;
        ext_frame_type : IN std_logic_vector(7 downto 0);
        TPM_tx_data : IN std_logic_vector(31 downto 0);
        TPM_write_to_fifo : IN std_logic;

        TPM_ready : IN std_logic;

    clear_ack: in std_logic;

        read_from_fifo_mux : IN std_logic;
        rx_over_mux : OUT std_logic;
        frame_type_mux : OUT std_logic_vector(7 downto 0);
        dout32_present_n_mux : OUT std_logic;
        dout32_mux : OUT std_logic_vector(31 downto 0);

        change_happened : OUT std_logic;
        RF_mode : OUT std_logic
        );
END COMPONENT;
signal      TX_ready: std_logic;
signal      ext_OK :    std_logic;
```

```
        signal      ext_frame_type :      std_logic_vector(7 downto 0);
        signal      TPM_tx_data :      std_logic_vector(31 downto 0);
        signal      TPM_write_to_fifo :      std_logic;

        signal      TPM_ready :      std_logic;
        signal      read_from_fifo_mux :      std_logic;
        signal      rx_over_mux :      std_logic;
        signal      frame_type_mux :      std_logic_vector(7 downto 0);
        signal      dout32_present_n_mux :      std_logic;
        signal      dout32_mux :      std_logic_vector(31 downto 0);

        signal      change_happened :      std_logic;
        signal      RF_mode :      std_logic;

signal clear_ack: std_logic;

--HF
        COMPONENT top_data_control
        PORT(
                clk : IN std_logic;
                rst : IN std_logic;
                reconfig : IN std_logic;
                cpu_cmd : IN std_logic_vector(3 downto 0);
                tpm_empty : IN std_logic;
                fifo_empty : IN std_logic;
                tpm_data_in : IN std_logic_vector(31 downto 0);
                zm_data_in : IN std_logic_vector(31 downto 0);
                re_tpm : OUT std_logic;
                wr_tpm : OUT std_logic;
                re_fifo : OUT std_logic;
                wr_fifo : OUT std_logic;
                io_finish : OUT std_logic;
                verify_finish : OUT std_logic;
                ready : OUT std_logic;
                tpm_data_out : OUT std_logic_vector(31 downto 0);
                zm_data_out : OUT std_logic_vector(31 downto 0)
                );
        END COMPONENT;


--HF
        COMPONENT cpu_top
        PORT(
                clk : IN std_logic;
```

```
        rst : IN std_logic;
        zm_change_mode_en : IN std_logic;
        zm_mode : IN std_logic;
        tpm_en : IN std_logic;
        con_ready : IN std_logic;
        zm_ready : IN std_logic;
        io_ready : IN std_logic;
        verify_result : IN std_logic_vector(1 downto 0);
        from_zm_en : IN std_logic;
        from_zm_cmd : IN std_logic_vector(7 downto 0);
        oneway_finish : IN std_logic;
        to_zm_en : OUT std_logic;
        to_zm_cmd : OUT std_logic_vector(7 downto 0);
        to_tpm_cmd : OUT std_logic_vector(3 downto 0);
        to_con_cmd : OUT std_logic_vector(3 downto 0);
        cpu_ready : OUT std_logic;
        tpm_change_mode_en : OUT std_logic;
        tpm_mode : OUT std_logic;
        tesla_init_check : out STD_LOGIC;
        idj_check1 : out STD_LOGIC;
        con_reconfig : OUT std_logic
        );
END COMPONENT;

----zzy
COMPONENT tpm
PORT(
        clk : IN std_logic;
        reset : IN std_logic;
        sta : IN std_logic_vector(3 downto 0);
        clear : IN std_logic;
        change : IN std_logic;
        center : IN std_logic;
        IO_finish : IN std_logic;
        we : IN std_logic;
        re : IN std_logic;
        din : IN std_logic_vector(31 downto 0);
        empty : OUT std_logic;
        full : OUT std_logic;
        over : OUT std_logic;
        compare_suc : OUT std_logic_vector(1 downto 0);
        clear_ack : OUT std_logic;
        IO_ready : OUT std_logic;
        re_en : OUT std_logic;
```

```
            Tpm_En : OUT std_logic;
            dout : OUT std_logic_vector(31 downto 0)
            );
    END COMPONENT;
    --
    signal cmd : std_logic_vector (3 downto 0);
    signal
clear,change,node_type,tpm_datafinish,tpm_we,tpm_re,tpm_empty,tpm_full,Kdisbute_end,
                tpm_IO_ready,tpm_in,Tpm_En : std_logic;

    signal compare_suc : std_logic_vector (1 downto 0);
    signal tpm_din ,tpm_dout : std_logic_vector( 31 downto 0);
    signal control_ready : std_logic;
    signal verify_finish_mem : std_logic;
    signal to_con_cmd_cpu : std_logic_vector(3 downto 0);
    signal reconfig_cpu : std_logic;

begin

    Inst_DCM: DCM_10 PORT MAP(
        CLKIN_IN => clk_in,
        RST_IN => reset,
        CLK => clk,
        CLKIN_IBUFG_OUT => open,
        CLK0_OUT => open,
        LOCKED_OUT => locked
    );
    rst <= not locked;


--cui
    Inst_top_peripheral: top_peripheral PORT MAP(
      --ext ports
        clk => clk,
        rst => rst,

        --for check
        check_rx_over => check_rx_over,

        rotary_a => rotary_a,
        rotary_b => rotary_b,
        clear_display => clear_display,
        ps2_clk => ps2_clk,
        ps2_data => ps2_data,
```

```
            lcd_d => lcd_d,
            lcd_rs => lcd_rs,
            lcd_rw =>lcd_rw ,
            lcd_e => lcd_e,
            strataflash_oe => strataflash_oe,
            strataflash_ce => strataflash_ce,
            strataflash_we => strataflash_we,
            RS232_rx => RS232_rx,
            RS232_tx => RS232_tx,
            RF_serial_rx => RF_serial_rx,
            RF_serial_tx =>RF_serial_tx ,
            frame_leds => frame_leds,

            --inner ports
            TPM_ready => TPM_ready,

          clear_ack => clear_ack,

            rx_over_mux => rx_over_mux,
            frame_type_mux => frame_type_mux,
            read_from_fifo_mux => read_from_fifo_mux,
            dout32_present_n_mux => dout32_present_n_mux,
            dout32_mux => dout32_mux,

            TX_ready => TX_ready,

            ext_OK => ext_OK,
            ext_frame_type => ext_frame_type,
            TPM_tx_data => TPM_tx_data,
            TPM_write_to_fifo => TPM_write_to_fifo,

            change_happened => change_happened,
            RF_mode => RF_mode
        );
--
--logic for abve module
          data32_present <= dout32_present_n_mux;
        TPM_ready_check <= TPM_ready;


--HF
        Inst_top_data_control: top_data_control PORT MAP(
            clk => clk,
            rst => rst,
```

```
--top_peripheral interconnect below
            --top_peripheral rx for TPM to read
            re_fifo => read_from_fifo_mux,
            fifo_empty => dout32_present_n_mux,
            zm_data_in => dout32_mux,

            --top_peripheral tx for TPM to write
            zm_data_out => TPM_tx_data,
            wr_fifo => TPM_write_to_fifo,
--top_peripheral interconnect above

            reconfig => reconfig_cpu,
            cpu_cmd => to_con_cmd_cpu,
            tpm_empty => tpm_empty,
            re_tpm => tpm_re,
            wr_tpm => tpm_we,
            io_finish => tpm_datafinish,
            verify_finish => verify_finish_mem,
            ready => control_ready,
            tpm_data_in => tpm_dout,
            tpm_data_out => tpm_din
      );
--
--logic for above module

--
      Inst_cpu_top: cpu_top PORT MAP(
            clk => clk,
            rst => rst,

--top_peripheral interconnect below
            --top_peripheral rx for TPM to read
            cpu_ready => TPM_ready,
            from_zm_en => rx_over_mux,
            from_zm_cmd => frame_type_mux,

            --top_peripheral tx for TPM to write
            zm_ready => TX_ready,
            to_zm_en => ext_OK,
            to_zm_cmd => ext_frame_type,

          --top_peripheral for reconfig and IDj
            zm_change_mode_en => change_happened,
```

```
            zm_mode => RF_mode,
--top_peripheral interconnect above

            tpm_en => Tpm_En,
            con_ready => control_ready,
            io_ready => tpm_IO_ready,
            verify_result => compare_suc,
            oneway_finish => verify_finish_mem,
            to_tpm_cmd => cmd,
            to_con_cmd => to_con_cmd_cpu,
            tpm_change_mode_en => change,
            tpm_mode => node_type,
            tesla_init_check =>open ,
            idj_check1 =>open ,
            con_reconfig => reconfig_cpu
      );
--
--logic for above module

--zzy
      Inst_tpm: tpm PORT MAP(
            clk => clk,
            reset => rst,
            sta => cmd,
            clear => TPM_clear,----IO
            change => change,----base change to center
            center => node_type,----sel base type or center type
            IO_finish => tpm_datafinish,
            we => tpm_we,
            re => tpm_re,
            din => tpm_din,
            empty => tpm_empty,
            full => tpm_full,
            over => Kdisbute_end,
            compare_suc => compare_suc,
            clear_ack => clear_ack,
            IO_ready => tpm_IO_ready,
            re_en => tpm_in,---condition of tpm data transport
            Tpm_En => Tpm_En,
            dout => tpm_dout
      );
      Tpm_En_check <= Tpm_En;

end Behavioral;
```

--中心控制器

--- cpu_pack.vhd

```
package cpu_pack is

   constant DATA_WIDTH                 : INTEGER := 32;
   constant CONTROL_CMD_WIDTH    : INTEGER := 4;
   CONSTANT TPM_CMD_WIDTH           : INTEGER := 4;
   CONSTANT ZM_CMD_WIDTH                 : INTEGER := 8;

   -- 控制 TPM 的命令
   -- INVALID COMMAND
   CONSTANT INVALID_TPM_CMD                    :
      STD_LOGIC_VECTOR(TPM_CMD_WIDTH-1 DOWNTO 0) := "0000";
   -- 初始化中的命令，公用的
   CONSTANT BC_REFRSH_K_Q_CMD                  :
      STD_LOGIC_VECTOR(TPM_CMD_WIDTH-1 DOWNTO 0) := "0001";
   CONSTANT BC_REFRSH_K_I_P_CMD                :
      STD_LOGIC_VECTOR(TPM_CMD_WIDTH-1 DOWNTO 0) := "0010";
   CONSTANT BC_REFRSH_K_O_CMD                  :
      STD_LOGIC_VECTOR(TPM_CMD_WIDTH-1 DOWNTO 0) := "0100";
   CONSTANT BC_REFRSH_K_Tn_CMD                 :
      STD_LOGIC_VECTOR(TPM_CMD_WIDTH-1 DOWNTO 0) := "1000";

   -- 基站
   -- ENCRYPTION COMMAND
   -- 用子网通信密钥加密
   CONSTANT B_ENC_K_I_CMD                      :
STD_LOGIC_VECTOR(TPM_CMD_WIDTH-1 DOWNTO 0) := "0001";
   --用与另一基站的通信密?
   CONSTANT B_ENC_K_P_CMD                      :
STD_LOGIC_VECTOR(TPM_CMD_WIDTH-1 DOWNTO 0) := "0010";
   --与第三基站的通信密钥
   CONSTANT B_ENC_K_PP_CMD                :
STD_LOGIC_VECTOR(TPM_CMD_WIDTH-1 DOWNTO 0) := "1110";
   --DECRYPTION COMMAND
   CONSTANT B_DEC_K_I_CMD                      :
STD_LOGIC_VECTOR(TPM_CMD_WIDTH-1 DOWNTO 0) := "0011";
   CONSTANT B_DEC_K_P_CMD                      :
STD_LOGIC_VECTOR(TPM_CMD_WIDTH-1 DOWNTO 0) := "0100";
   CONSTANT B_DEC_K_PP_CMD                     :
```

```
STD_LOGIC_VECTOR(TPM_CMD_WIDTH-1 DOWNTO 0) := "1101";
   CONSTANT B_DEC_K_ZI_CMD                    :
STD_LOGIC_VECTOR(TPM_CMD_WIDTH-1 DOWNTO 0) := "0110";
   -- HASH
   CONSTANT B_HASH_CMD                        :
STD_LOGIC_VECTOR(TPM_CMD_WIDTH-1 DOWNTO 0) := "0101";
   --  密钥分配
   CONSTANT B_DISPATCH_CMD                    :
STD_LOGIC_VECTOR(TPM_CMD_WIDTH-1 DOWNTO 0) := "0111";
   --  节点加入
   CONSTANT B_NODE_JOIN_CMD                   :
STD_LOGIC_VECTOR(TPM_CMD_WIDTH-1 DOWNTO 0) := "1000";

   --密钥更新
   CONSTANT B_FRESH_K_I_CMD                   :
STD_LOGIC_VECTOR(TPM_CMD_WIDTH-1 DOWNTO 0) := "1001";
   CONSTANT B_FRESH_K_I_P_CMD                 :
STD_LOGIC_VECTOR(TPM_CMD_WIDTH-1 DOWNTO 0) := "1010";
   CONSTANT B_FRESH_K_I_PP_CMD                :
STD_LOGIC_VECTOR(TPM_CMD_WIDTH-1 DOWNTO 0) := "1100";
   CONSTANT B_FRESH_K_T_I_CMD                 :
STD_LOGIC_VECTOR(TPM_CMD_WIDTH-1 DOWNTO 0) := "1011";

   --  组建  TESLA  密钥包
   CONSTANT B_MAKE_TESLA_CMD                  :
STD_LOGIC_VECTOR(TPM_CMD_WIDTH-1 DOWNTO 0) := "0111";

   --  节点
   -- ENCRYPTION COMMAND
   --  用子网通信密钥加密
   CONSTANT C_ENC_K_I_CMD                     :
STD_LOGIC_VECTOR(TPM_CMD_WIDTH-1 DOWNTO 0) := "0001";
   --  接受 TESLA  初始包后更新    K_To,    Ko
   CONSTANT C_REFRESH_TESLA_INIT              :
STD_LOGIC_VECTOR(TPM_CMD_WIDTH-1 DOWNTO 0) := "1001";
   --用  K 子  （加入新的子网）  解密
   CONSTANT C_ENC_K_P_CMD                     :
STD_LOGIC_VECTOR(TPM_CMD_WIDTH-1 DOWNTO 0) := "0110";
   --加入新子网重新计算  K 子
   CONSTANT C_CAL_K_ZI_CMD                    :
STD_LOGIC_VECTOR(TPM_CMD_WIDTH-1 DOWNTO 0) := "1000";
   --单向  K<T_i-1>
   CONSTANT C_ONEWAY_K_T_I_CMD                :
STD_LOGIC_VECTOR(TPM_CMD_WIDTH-1 DOWNTO 0) := "0010";
```

```
   -- 验证 K<T_i-1>
   CONSTANT C_VERIFY_K_T_I_CMD                    :
STD_LOGIC_VECTOR(TPM_CMD_WIDTH-1 DOWNTO 0) := "1010";
   -- ?K<T_i-1>,错误不更新
   --CONSTANT C_UNFRESH_K_T_I_CMD                  :
STD_LOGIC_VECTOR(TPM_CMD_WIDTH-1 DOWNTO 0) := "1011";
   -- 解密 tesla 密钥包信息
   CONSTANT C_DEC_TESLA_CMD                       :
STD_LOGIC_VECTOR(TPM_CMD_WIDTH-1 DOWNTO 0) := "0100";
   -- HASH
   CONSTANT C_HASN_CMD                            :
STD_LOGIC_VECTOR(TPM_CMD_WIDTH-1 DOWNTO 0) := "0011";
   -- 解密基站发过来的消息
   CONSTANT C_DEC_TESLA_INIT_CMD                  :
STD_LOGIC_VECTOR(TPM_CMD_WIDTH-1 DOWNTO 0) := "0101";

   -- 控制 data_control 的命令
   -- INVALID CMD
   CONSTANT INVALID_CON_CMD:  STD_LOGIC_VECTOR(CONTROL_CMD_WIDTH-1
downto 0) := "0000";
   -- 从 TPM 读正常的数据 写入 FIFO
   constant READ_DATA        :      STD_LOGIC_VECTOR(CONTROL_CMD_WIDTH-1
downto 0) := "0001";
   -- 将数据从 FIFO 写入 TPM
   constant WRITE_DATA   :  STD_LOGIC_VECTOR(CONTROL_CMD_WIDTH-1   downto
0) := "0010";

   -- 把 TPM 读入 FIFO2，主要暂存解密完成的数据
   CONSTANT                    READ_DATA_TO_FIFO2                    :
STD_LOGIC_VECTOR(CONTROL_CMD_WIDTH-1 downto 0) := "0011";
  -- 从 fifo2 中把数据读入 TPM
  CONSTANT                    WRITE_DATA_FROM_FIFO2                 :
STD_LOGIC_VECTOR(CONTROL_CMD_WIDTH-1 downto 0) := "0100";

    --  从 TPM 读解密后的 TESLA 初始包
   constant READ_TESLA_INIT      :  STD_LOGIC_VECTOR(CONTROL_CMD_WIDTH-1
downto 0) := "0101";
   --  把从 TPM 读取的解密后的 TESLA 初始包的内容分开后送回到 TPM
   constant WRITE_TESLA_INIT     :  STD_LOGIC_VECTOR(CONTROL_CMD_WIDTH-1
downto 0) := "0110";

   -- 从 TPM 读 解密后的 TESLA 密钥包
   constant READ_TESLA   :  STD_LOGIC_VECTOR(CONTROL_CMD_WIDTH-1   downto
0) := "0111";
```

```
   -- 组建 TESLA 密钥包需要 i 值
   constant WRITE_I          :  STD_LOGIC_VECTOR(CONTROL_CMD_WIDTH-1  downto
0) := "1000";

   -- 把 K<Ti-1> 送入 TPM
   constant WRITE_K_T          :       STD_LOGIC_VECTOR(CONTROL_CMD_WIDTH-1
downto 0) := "1001";
   -- 从 TPM 读做了 单向函数的 K<Ti-1>
   constant READ_K_T       :  STD_LOGIC_VECTOR(CONTROL_CMD_WIDTH-1  downto
0) := "1010";

   -- 把验证正确后的 K_Ti 送给 TPM
   constant WRTIT_NEW_K_T : STD_LOGIC_VECTOR(CONTROL_CMD_WIDTH-1 downto
0) := "1011";

   -- 把 E（( )）送入 TPM
   constant WRITE_E_K_I   :  STD_LOGIC_VECTOR(CONTROL_CMD_WIDTH-1  downto
0):= "1100";

 -- 把 TPM 里面的无效数据读空
 CONSTANT READ_INVALID_DATA_CMD          :
STD_LOGIC_VECTOR(CONTROL_CMD_WIDTH-1 downto 0) := "1101";
 -- 往 TPM 中写空的数据
 constant WRITE_NULL_CMD                 :
STD_LOGIC_VECTOR(CONTROL_CMD_WIDTH-1 downto 0) := "1110";

   -- 送给 ZM 的命令
   -- INVALID_ZM_CMD
   CONSTANT INVALID_ZM_CMD                          :
STD_LOGIC_VECTOR(ZM_CMD_WIDTH-1 DOWNTO 0) := (others => '0');
   -- 注入密钥
   constant SND_K_Q_CMD          :          STD_LOGIC_VECTOR(ZM_CMD_WIDTH-1
DOWNTO 0) := "00000001";
   constant SND_K_I_P_CMD        :          STD_LOGIC_VECTOR(ZM_CMD_WIDTH-1
DOWNTO 0) := "00000010";
   constant SND_K_O_CMD          :          STD_LOGIC_VECTOR(ZM_CMD_WIDTH-1
DOWNTO 0) := "00000011";
   constant SND_K_Tn_CMD               :    STD_LOGIC_VECTOR(ZM_CMD_WIDTH-1
DOWNTO 0) := "00000100";
   --⑺透  硪桓基站的命令
   CONSTANT SEND_TO_COMPUTER   :    STD_LOGIC_VECTOR(ZM_CMD_WIDTH-1
DOWNTO 0) := "00000101";
   -- 基站发给终端节点
   CONSTANT SEND_TESLA_CMD             :
```

```
STD_LOGIC_VECTOR(ZM_CMD_WIDTH-1 DOWNTO 0) := "00000110";
  CONSTANT  SEND_TESLA_INIT_CMD   : STD_LOGIC_VECTOR(ZM_CMD_WIDTH-1
DOWNTO 0) := "00000111";
  -- 终端发给基站
  CONSTANT SND_DATA_CMD                :
STD_LOGIC_VECTOR(ZM_CMD_WIDTH-1 DOWNTO 0) := "00001000";

  ----- SEND TO SRCEEN
  CONSTANT  SEND_TO_SCREEN          : STD_LOGIC_VECTOR(ZM_CMD_WIDTH-1
DOWNTO 0) := "00000110";

  -- 从 ZM 接受的命令
  -- 接受到初始化密钥
  constant REV_K_Q_CMD          :          STD_LOGIC_VECTOR(ZM_CMD_WIDTH-1
DOWNTO 0) := "00000001";
  constant REV_K_I_P_CMD        :          STD_LOGIC_VECTOR(ZM_CMD_WIDTH-1
DOWNTO 0) := "00000010";
  constant REV_K_O_CMD          :          STD_LOGIC_VECTOR(ZM_CMD_WIDTH-1
DOWNTO 0) := "00000011";
  constant REV_K_Tn_CMD             :      STD_LOGIC_VECTOR(ZM_CMD_WIDTH-1
DOWNTO 0) := "00000100";
  -- 从另外一个基站处接受到数据
  constant REV_FORM_BSTATION2 :            STD_LOGIC_VECTOR(ZM_CMD_WIDTH-1
DOWNTO 0) := "00000101";
  -- 终端节点收到基站发来的数据
  constant REV_TESLA_INIT_CMD  :           STD_LOGIC_VECTOR(ZM_CMD_WIDTH-1
DOWNTO 0) := "00000111";
  constant REV_TESLA_CMD        :          STD_LOGIC_VECTOR(ZM_CMD_WIDTH-1
DOWNTO 0) := "00000110";
  -- 基站接受到了节点发来的消息，需要解密，然后加密发送给另外一个基站
  constant DEC_ENC_SND_CMD      :          STD_LOGIC_VECTOR(ZM_CMD_WIDTH-1
DOWNTO 0) := "00001000";

  --  节点接受到 PS2 的数据，然后加密发送给基站
  constant ENC_SND_DATA_CMD         :      STD_LOGIC_VECTOR(ZM_CMD_WIDTH-1
DOWNTO 0) := "00001001";



  --其他一些常数定义
  CONSTANT VERIFY_WIDTH :     INTEGER := 2;
  CONSTANT VERIFY_SUC    :    STD_LOGIC_VECTOR(VERIFY_WIDTH-1  DOWNTO
0) := "10";
  CONSTANT VERIFY_FAIL    :   STD_LOGIC_VECTOR(VERIFY_WIDTH-1  DOWNTO
```

```
0) := "01";
   CONSTANT HASH_WIDTH :  INTEGER := 2;
   CONSTANT HASH_SUC :      STD_LOGIC_VECTOR(VERIFY_WIDTH-1 DOWNTO 0) :=
"10";
   CONSTANT HASH_FAIL :      STD_LOGIC_VECTOR(VERIFY_WIDTH-1 DOWNTO 0) :=
"01";
   CONSTANT COUNTER_WIDTH : INTEGER := 36;
--      1 miniute
   CONSTANT DISPATCH_TIME : STD_LOGIC_VECTOR(COUNTER_WIDTH-1 DOWNTO
0) := X"100000000";
     --10 us
   --CONSTANT      DISPATCH_TIME      :      STD_LOGIC_VECTOR(COUNTER_WIDTH-1
DOWNTO 0) := X"000000200";
   CONSTANT DEF_ZERO          : STD_LOGIC_VECTOR(COUNTER_WIDTH-1 DOWNTO
0) := (others => '0');

end cpu_pack;
```

--- cpu_top.vhd

```
entity cpu_top is
      Port ( clk : in    STD_LOGIC;
                rst : in    STD_LOGIC;
                zm_change_mode_en : in STD_LOGIC;
                zm_mode                  : in STD_LOGIC;
                tpm_en : in std_logic;
                con_ready : in    STD_LOGIC;
                zm_ready : in    STD_LOGIC;
                io_ready : in std_logic;
                verify_result : in    STD_LOGIC_VECTOR (VERIFY_WIDTH-1 downto 0);
                from_zm_en : in    STD_LOGIC;
                from_zm_cmd : in    STD_LOGIC_VECTOR (ZM_CMD_WIDTH-1 downto 0);
                oneway_finish      : in    std_logic;
                to_zm_en : out    STD_LOGIC;
                to_zm_cmd : out    STD_LOGIC_VECTOR (ZM_CMD_WIDTH-1 downto 0);
                to_tpm_cmd : out    STD_LOGIC_VECTOR (TPM_CMD_WIDTH-1 downto 0);
                to_con_cmd : out    STD_LOGIC_VECTOR (CONTROL_CMD_WIDTH-1 downto
0);
                cpu_ready : out    STD_LOGIC;
                tpm_change_mode_en : out STD_LOGIC;
                tpm_mode                  : out STD_LOGIC;
                 tesla_init_check : out STD_LOGIC;
                 idj_check1 : out STD_LOGIC;
                con_reconfig              : out std_logic         -- use to reconfig control_data unit, ‘1’
```

```
                    ----保持一个周期 激发重构
                );
end cpu_top;

architecture RTL of cpu_top is

type state is( init_state1, wait_state1, wait_state2, reconfig_state1, reconfig_state2);
signal curr_state, next_state : state;
signal BP_tpm_change_mode_en_top, BP_tpm_mode_top : std_logic;
signal init_en, b_en, c_en : std_logic;

signal BP_cpu_ready_top, BP_cpu_ready_init, BP_cpu_ready_base, BP_cpu_ready_client : std_logic;
signal BP_to_zm_en_base, BP_to_zm_en_client : std_logic;

signal BP_to_con_cmd_init, BP_to_con_cmd_base, BP_to_con_cmd_client : STD_LOGIC_VECTOR (CONTROL_CMD_WIDTH-1 downto 0);
signal BP_to_zm_cmd_base, BP_to_zm_cmd_client : STD_LOGIC_VECTOR (ZM_CMD_WIDTH-1 downto 0);
signal BP_to_tpm_cmd_base, BP_to_tpm_cmd_client : STD_LOGIC_VECTOR (TPM_CMD_WIDTH-1 downto 0);

signal BP_time_begin, BP_time_end : std_logic;

    COMPONENT cpu_init
    PORT(
        clk : IN std_logic;
        rst : IN std_logic;
        con_ready : in std_logic;
        from_zm_en : IN std_logic;
        from_zm_cmd : IN std_logic_vector(7 downto 0);
        to_con_cmd : OUT std_logic_vector(3 downto 0);
        cpu_ready : OUT std_logic
        );
    END COMPONENT;

    COMPONENT cpu_base
    PORT(
        clk : IN std_logic;
        rst : IN std_logic;
        b_en : IN std_logic;
        tpm_en : IN std_logic;
        io_ready : IN std_logic;
        hash_result : IN std_logic_vector(1 downto 0);
```

```
        from_zm_en : IN std_logic;
        from_zm_cmd : IN std_logic_vector(7 downto 0);
        zm_ready : IN std_logic;
        con_ready : in std_logic;
        time_end : IN std_logic;
        to_zm_en : OUT std_logic;
        to_zm_cmd : OUT std_logic_vector(7 downto 0);
        to_tpm_cmd : OUT std_logic_vector(3 downto 0);
        to_con_cmd : OUT std_logic_vector(3 downto 0);
        time_begin : OUT std_logic;
        -- check
        tesla_init_check : out STD_LOGIC;
        idj_check1 : out STD_LOGIC;

        cpu_ready : OUT std_logic
        );
END COMPONENT;

COMPONENT cpu_base_counter
PORT(
        clk : IN std_logic;
        rst : IN std_logic;
        en : IN std_logic;
        time_end : OUT std_logic
        );
END COMPONENT;

COMPONENT cpu_client
PORT(
        clk : IN std_logic;
        rst : IN std_logic;
        c_en : IN std_logic;
        tpm_en : IN std_logic;
        io_ready : IN std_logic;
        verify_result : IN std_logic_vector(1 downto 0);
        oneway_finish : IN std_logic;
        from_zm_en : IN std_logic;
        from_zm_cmd : IN std_logic_vector(7 downto 0);
        zm_ready : IN std_logic;
        con_ready : in std_logic;
        to_zm_en : OUT std_logic;
        to_zm_cmd : OUT std_logic_vector(7 downto 0);
        to_tpm_cmd : OUT std_logic_vector(3 downto 0);
        to_con_cmd : OUT std_logic_vector(3 downto 0);
```

```
        cpu_ready : OUT std_logic
        );
    END COMPONENT;



begin
    Inst_cpu_init: cpu_init PORT MAP(
        clk => clk ,
        rst => rst ,
        con_ready => con_ready,
        to_con_cmd => BP_to_con_cmd_init,
        cpu_ready => BP_cpu_ready_init,
        from_zm_en => from_zm_en ,
        from_zm_cmd => from_zm_cmd
    );

    Inst_cpu_base: cpu_base PORT MAP(
        clk => clk ,
        rst => rst ,
        b_en => b_en ,
        tpm_en => tpm_en ,
        io_ready => io_ready ,
        hash_result => verify_result ,
        from_zm_en => from_zm_en ,
        from_zm_cmd => from_zm_cmd ,
        to_zm_en => BP_to_zm_en_base,
        to_zm_cmd => BP_to_zm_cmd_base,
        to_tpm_cmd => BP_to_tpm_cmd_base,
        to_con_cmd => BP_to_con_cmd_base,
        zm_ready => zm_ready ,
        con_ready => con_ready,
        time_end => BP_time_end ,
        time_begin => BP_time_begin ,
        tesla_init_check => tesla_init_check,
        idj_check1 => idj_check1,
        cpu_ready => BP_cpu_ready_base
    );

    Inst_cpu_base_counter: cpu_base_counter PORT MAP(
        clk => clk ,
        rst => rst ,
        en => BP_time_begin,
        time_end => BP_time_end
```

```
    );

    Inst_cpu_client: cpu_client PORT MAP(
        clk => clk ,
        rst => rst ,
        c_en => c_en,
        tpm_en => tpm_en ,
        io_ready => io_ready ,
        verify_result => verify_result,
        oneway_finish => oneway_finish,
        from_zm_en => from_zm_en,
        from_zm_cmd => from_zm_cmd ,
        to_zm_en => BP_to_zm_en_client,
        to_zm_cmd => BP_to_zm_cmd_client,
        to_tpm_cmd => BP_to_tpm_cmd_client,
        to_con_cmd => BP_to_con_cmd_client,
        zm_ready => zm_ready ,
        con_ready => con_ready,
        cpu_ready => BP_cpu_ready_client
    );

PROCESS(clk, rst)
begin
    if rst = '1' then
            curr_state <= init_state1;
        elsif clk'event and clk='1' then
            curr_state <= next_state;

            ------------------------------------------------------------
            if(init_en = '1')then
                to_tpm_cmd <=   INVALID_TPM_CMD;
                to_con_cmd <=   BP_to_con_cmd_init;
                to_zm_en <= '0';
                to_zm_cmd <= INVALID_ZM_CMD;
            elsif(c_en = '1')then
                to_tpm_cmd <=   BP_to_tpm_cmd_client;
                to_con_cmd <=   BP_to_con_cmd_client;
                to_zm_en <= BP_to_zm_en_client;
                to_zm_cmd <= BP_to_zm_cmd_client;
            elsif(b_en = '1')then
                to_tpm_cmd <=   BP_to_tpm_cmd_base;
                to_con_cmd <=   BP_to_con_cmd_base;
                to_zm_en <= BP_to_zm_en_base;
                to_zm_cmd <= BP_to_zm_cmd_base;
```

```
                else
                    to_tpm_cmd <=    INVALID_TPM_CMD;
                    to_con_cmd <=    INVALID_CON_CMD;
                    to_zm_en <= '0';
                    to_zm_cmd <= INVALID_ZM_CMD;
                end if;
                cpu_ready <= BP_cpu_ready_top and BP_cpu_ready_init and BP_cpu_ready_base
and BP_cpu_ready_client;
                tpm_change_mode_en <= BP_tpm_change_mode_en_top;
                tpm_mode       <= BP_tpm_mode_top;
            end if;
end process;

PROCESS(curr_state, zm_change_mode_en, zm_mode )
begin

    case curr_state is
    --------------------------------------------------------------------------------
    -- 初始化状态
    --------------------------------------------------------------------------------
    when init_state1 =>
        if(zm_change_mode_en = '1')then
            BP_cpu_ready_top <= '0';
            con_reconfig <= '1';
            if (zm_mode = '1') then
                init_en <= '0';
                b_en <= '0';
                c_en <= '0';
                BP_tpm_change_mode_en_top <= '1';
                BP_tpm_mode_top <= '1';
                next_state <= reconfig_state1;
            else
                init_en <= '0';
                b_en <= '0';
                c_en <= '0';
                BP_tpm_change_mode_en_top <= '1';
                BP_tpm_mode_top <= '0';
                next_state <= reconfig_state2;
            end if;
        else
            con_reconfig <= '0';
            BP_cpu_ready_top <= '1';
            BP_tpm_change_mode_en_top <= '0';
            BP_tpm_mode_top <= '0';
```

```
            BP_cpu_ready_top <= '1';
            init_en <= '1';
            b_en <= '0';
            c_en <= '0';
            next_state <= init_state1;
        end if;
-------------------------------------------------------------------------------
-- 等  刺?
--wait_state1, base station wait state
--wait_state2, client station wait state
-------------------------------------------------------------------------------
when wait_state1 =>
        -- 如果需要重构的话
        if(zm_change_mode_en = '1')then
            con_reconfig <= '1';
            BP_cpu_ready_top <= '0';
            if (zm_mode = '1') then
                init_en <= '0';
                b_en <= '0';
                c_en <= '0';
                BP_tpm_change_mode_en_top <= '1';
                BP_tpm_mode_top <= '1';
                next_state <= reconfig_state1;
            else
                init_en <= '0';
                b_en <= '0';
                c_en <= '0';
                BP_tpm_change_mode_en_top <= '1';
                BP_tpm_mode_top <= '0';
                next_state <= reconfig_state2;
            end if;
        else
            con_reconfig <= '0';
            BP_tpm_change_mode_en_top <= '0';
            BP_tpm_mode_top <= '1';
            BP_cpu_ready_top <= '1';
            init_en <= '0';
            b_en <= '1';
            c_en <= '0';
            next_state <= wait_state1;
        end if;
when wait_state2 =>
        -- 如果需要重构的话
        if(zm_change_mode_en = '1')then
```

```
            con_reconfig <= '1';
            BP_cpu_ready_top <= '0';
            if (zm_mode = '1') then
                init_en <= '0';
                b_en <= '0';
                c_en <= '0';
                BP_tpm_change_mode_en_top <= '1';
                BP_tpm_mode_top <= '1';
                next_state <= reconfig_state1;
            else
                init_en <= '0';
                b_en <= '0';
                c_en <= '0';
                BP_tpm_change_mode_en_top <= '1';
                BP_tpm_mode_top <= '0';
                next_state <= reconfig_state2;
            end if;
        else
            con_reconfig <= '0';
            BP_tpm_change_mode_en_top <= '0';
            BP_tpm_mode_top <= '0';
            BP_cpu_ready_top <= '1';
            init_en <= '0';
            b_en <= '0';
            c_en <= '1';
            next_state <= wait_state2;
        end if;
-------------------------------------------------------------------------------
--  重构状态, reconfig1 -- base, reconfig2 -- client =---
-------------------------------------------------------------------------------
when reconfig_state1 =>
        con_reconfig <= '1';
        BP_tpm_change_mode_en_top <= '0';
        BP_tpm_mode_top <= '1';
        BP_cpu_ready_top <= '0';
        init_en <= '0';
        b_en <= '1';
        c_en <= '0';
        next_state <= wait_state1;
when reconfig_state2 =>
        con_reconfig <= '1';
        BP_tpm_change_mode_en_top <= '0';
        BP_tpm_mode_top <= '0';
        BP_cpu_ready_top <= '0';
```

```
            init_en <= '0';
            b_en <= '0';
            c_en <= '1';
            next_state <= wait_state2;
        end case;

end PROCESS;
end RTL;
```

--- cpu_init.vhd

```
entity cpu_init is
        Port ( clk : in    STD_LOGIC;
                rst : in    STD_LOGIC;
                from_zm_en : in    STD_LOGIC;
                from_zm_cmd : in    STD_LOGIC_VECTOR (ZM_CMD_WIDTH-1 downto 0);
                con_ready : in std_logic;
                to_con_cmd : out    STD_LOGIC_VECTOR (CONTROL_CMD_WIDTH-1 downto
0);
                cpu_ready : out    STD_LOGIC
              );
end cpu_init;

architecture RTL of cpu_init is

type state is( init_state2,
                        init_state5, init_state8, init_state11,
                        wait_state);
signal curr_state, next_state : state;
signal BP_to_con_cmd : STD_LOGIC_VECTOR (CONTROL_CMD_WIDTH-1 downto 0);
signal BP_cpu_ready : std_logic;

begin

PROCESS(clk, rst)
begin
            if rst = '1' then
                curr_state <= wait_state;
                to_con_cmd <= INVALID_CON_CMD;
                cpu_ready <='0';
            elsif clk'event and clk='1' then
                curr_state <= next_state;
                -------------------------------------------------
                to_con_cmd <= BP_to_con_cmd;
                cpu_ready <= BP_cpu_ready;
```

```
        end if;
end process;


process(curr_state, from_zm_en, from_zm_cmd, con_ready)

begin
    case curr_state is
    ------------------------------------------------------------------------------
    --  等待状态
    ------------------------------------------------------------------------------
    when wait_state =>

        BP_to_con_cmd <= INVALID_CON_CMD;
        if(from_zm_en = '1')then
            BP_cpu_ready <= '0';
            case from_zm_cmd is
            when REV_K_Q_CMD =>    next_state <= init_state2;
            when REV_K_I_P_CMD => next_state <= init_state5;
            when REV_K_O_CMD => next_state <= init_state8;
            when REV_K_Tn_CMD => next_state <= init_state11;
            when others => next_state <= wait_state;
            end case;
        else
            BP_cpu_ready <= '1';
            next_state <= wait_state;
        end if;
    ------------------------------------------------------------------------------
    --   初始状态，开始接受四个密钥  -- K 全
    ------------------------------------------------------------------------------
    when init_state2 =>
        BP_cpu_ready <= '0';
        --  通知  data_control  把    K 全  送入  TPM
        BP_to_con_cmd <= WRITE_DATA;
        next_state <= wait_state;
    ------------------------------------------------------------------------------
    --   初始状态，      -- Ki'（Ki"）
    ------------------------------------------------------------------------------
    when init_state5 =>
        BP_cpu_ready <= '0';
            --  通知  data_control  把    K?送入  TPM
            BP_to_con_cmd <= WRITE_DATA;
            next_state <= wait_state;
    ------------------------------------------------------------------------------
```

```
    --  初始状态，开始接受四个密钥 -- Ko
    ------------------------------------------------------------------------
    when init_state8 =>
        BP_cpu_ready <= '0';
            -- 通?data_control 把  K 全 送入 TPM
            BP_to_con_cmd <= WRITE_DATA;
            next_state <= wait_state;
    ------------------------------------------------------------------------
    --  初始状态，开始邮芩母雒钥 -- K_Tn
    ------------------------------------------------------------------------
    when init_state11 =>
        BP_cpu_ready <= '0';
            -- 口?data_control 把  K?送入 TPM
            BP_to_con_cmd <= WRITE_DATA;
            next_state <= wait_state;
    end case;
end process;

end RTL;
```

-- cpu_base.vhd

```
entity cpu_base is
    Port ( clk : in    STD_LOGIC;
           rst : in    STD_LOGIC;
           b_en : in    STD_LOGIC;
           tpm_en : in std_logic;
           io_ready : in std_logic;
           hash_result : in    STD_LOGIC_VECTOR (HASH_WIDTH-1 downto 0);
           from_zm_en : in    STD_LOGIC;
           from_zm_cmd : in    STD_LOGIC_VECTOR (ZM_CMD_WIDTH-1 downto 0);
           zm_ready : in    STD_LOGIC;
           con_ready : in    STD_LOGIC;
           time_end : in    STD_LOGIC;
           to_zm_en : out    STD_LOGIC;
           to_zm_cmd : out    STD_LOGIC_VECTOR (ZM_CMD_WIDTH-1 downto 0);
           to_tpm_cmd : out    STD_LOGIC_VECTOR (TPM_CMD_WIDTH-1 downto 0);
           to_con_cmd : out    STD_LOGIC_VECTOR (CONTROL_CMD_WIDTH-1 downto
0);
           time_begin : out STD_LOGIC;

            tesla_init_check : out STD_LOGIC;
```

```
                idj_check1 : out STD_LOGIC;
               cpu_ready : out    STD_LOGIC
                  );
end cpu_base;

architecture RTL of cpu_base is

type state is(init_state1, init_state2, init_state3, init_state4,
                        wait_state, fresh_k_i_state1,fresh_k_i_state2,
                        make_tesla_state1,        make_tesla_state2,        make_tesla_state3,
make_tesla_state4,
                        enc_dec_snd_state1,       enc_dec_snd_state2,       enc_dec_snd_state3,
enc_dec_snd_state4,
                        enc_dec_snd_state5, enc_dec_snd_state6, enc_dec_snd_state7,
                        hash_verify_state1, hash_verify_state2, hash_verify_state3);
signal curr_state,next_state : state;
signal BP_to_zm_en : std_logic;
signal BP_to_con_cmd : STD_LOGIC_VECTOR (CONTROL_CMD_WIDTH-1 downto 0);
signal BP_to_zm_cmd : STD_LOGIC_VECTOR (ZM_CMD_WIDTH-1 downto 0);
signal BP_to_tpm_cmd : STD_LOGIC_VECTOR (TPM_CMD_WIDTH-1 downto 0);
signal BP_cpu_ready : std_logic;
signal BP_time_begin : std_logic;

begin

PROCESS(clk, rst)
begin
        if rst = '1' then
            curr_state <= wait_state;
        elsif clk'event and clk='1' then
            curr_state <= next_state;
        end if;
end PROCESS;

            ---------------------------------------------
            to_tpm_cmd <= BP_to_tpm_cmd;
            to_con_cmd <= BP_to_con_cmd;
            to_zm_en <= BP_to_zm_en;
            to_zm_cmd <= BP_to_zm_cmd;
            cpu_ready <= BP_cpu_ready;
            time_begin <= BP_time_begin;

PROCESS(curr_state,  b_en,   hash_result,  from_zm_en,  from_zm_cmd,  zm_ready,  con_ready,
io_ready, tpm_en, time_end,
```

```
                BP_cpu_ready, BP_to_tpm_cmd, BP_to_zm_en, BP_to_con_cmd, BP_to_zm_cmd,
BP_to_tpm_cmd, BP_time_begin )


begin
if( b_en = '1' ) then
    case curr_state is
    ------------------------------------------------------------------------------
    -- ID 的处?   ------------------------------------------------------------------------------
    when init_state1 =>
        BP_time_begin <= '0';
        BP_cpu_ready <= '0';
        BP_to_tpm_cmd <= INVALID_TPM_CMD;
        BP_to_zm_en <= '0';
        BP_to_zm_cmd <= INVALID_ZM_CMD;
        tesla_init_check <= '0';
        idj_check1 <= '0';
        if(io_ready = '1')then
            -- 通知 data_control 把  ID 基 送入 TPM
            BP_to_con_cmd <= WRITE_DATA;
            next_state <= init_state2;
        else
            BP_to_con_cmd <= INVALID_CON_CMD;
            next_state <= init_state1;
        end if;
    ------------------------------------------------------------------------------
    -- TESlA 初始包的处理
    ------------------------------------------------------------------------------
    when init_state2 =>
        BP_time_begin <= '0';
        BP_cpu_ready <= '0';
        BP_to_tpm_cmd <= INVALID_TPM_CMD;
        BP_to_zm_en <= '0';
        BP_to_zm_cmd <= INVALID_ZM_CMD;
        tesla_init_check <= '0';
        idj_check1 <= '1';
        if( io_ready = '1' )then
            -- 通知 data_control 把 deta, To 送入 TPM
            BP_to_con_cmd <= WRITE_NULL_CMD;
            next_state <= init_state3;
        else
            BP_to_con_cmd <= INVALID_CON_CMD;
            next_state <= init_state2;
        end if;
```

```
when init_state3 =>
        BP_cpu_ready <= '0';
        BP_time_begin <= '0';
        BP_to_tpm_cmd <= INVALID_TPM_CMD;
        BP_to_zm_en <= '0';
        BP_to_zm_cmd <= INVALID_ZM_CMD;
        tesla_init_check <= '0';
        idj_check1 <= '1';
        if(io_ready = '1')then
            -- 通知 data_control 把 TESLA 初始包 送入 ZM
            BP_to_con_cmd <= READ_DATA;
            next_state <= init_state4;
        else
            BP_to_con_cmd <= INVALID_CON_CMD;
            next_state <= init_state3;
        end if;
when init_state4 =>
        BP_cpu_ready <= '0';
        BP_to_tpm_cmd <= INVALID_TPM_CMD;
        BP_to_con_cmd <= INVALID_CON_CMD;
        idj_check1 <= '1';
        if(tpm_en = '1')then
            -- 通知 ZM 把 TESLA 初始包 发送出去
            BP_to_zm_en <= '1';
            BP_to_zm_cmd <= SEND_TESLA_INIT_CMD;
            --使能计数器的信号
            BP_time_begin <= '1';
            tesla_init_check <= '1';
            next_state <= wait_state;
        else
            BP_to_zm_en <= '0';
            BP_to_zm_cmd <= INVALID_ZM_CMD;
            BP_time_begin <= '0';
            tesla_init_check <= '0';
            next_state <= init_state4;
        end if;
--------------------------------------------------------------------------------
-- 等待状态
--------------------------------------------------------------------------------
when wait_state =>
        BP_time_begin <= '1';
        BP_to_zm_en <= '0';
        BP_cpu_ready <= '1';
        BP_to_zm_cmd <= INVALID_ZM_CMD;
```

```
            BP_to_con_cmd <= INVALID_CON_CMD;
            tesla_init_check <= '1';
            idj_check1 <= '1';
            if(time_end = '1')then
                -- 分发密钥时间到
                BP_cpu_ready <= '0';
                BP_to_tpm_cmd <= B_FRESH_K_I_CMD;
                next_state <= fresh_k_i_state1;
            elsif(from_zm_en = '1')then
                BP_cpu_ready <= '0';
                BP_to_tpm_cmd <= INVALID_TPM_CMD;
                case(from_zm_cmd) is
                -- zm 有数据需要 解密，然后加密，发给另外的基站
                when DEC_ENC_SND_CMD =>
                    next_state <= enc_dec_snd_state1;
                when REV_FORM_BSTATION2 =>
                    next_state <= wait_state; ----------RECEIVE FROM
ANOTHER STATION----------
                when others =>
                    next_state <= wait_state;
                end case;
            else
                BP_cpu_ready <= '1';
                BP_to_tpm_cmd <= INVALID_TPM_CMD;
                next_state <= wait_state;
            end if;
        ------------------------------------------------------------------------
        -- TESlA 密钥包的处理
        ------------------------------------------------------------------------

        when fresh_k_i_state1 =>
            BP_cpu_ready <= '0';
            BP_time_begin <= '1';
            BP_to_zm_en <= '0';
            BP_to_zm_cmd <= INVALID_ZM_CMD;
            BP_to_tpm_cmd <= INVALID_TPM_CMD;
            BP_to_con_cmd <= INVALID_CON_CMD;
            tesla_init_check <= '1';
            idj_check1 <= '1';
            --if(tpm_en = '1')then
            next_state <= fresh_k_i_state2;
            --else
        when fresh_k_i_state2 =>
            BP_cpu_ready <= '0';
```

```
        BP_time_begin <= '1';
        BP_to_zm_en <= '0';
        BP_to_zm_cmd <= INVALID_ZM_CMD;
        BP_to_con_cmd <= INVALID_CON_CMD;
        tesla_init_check <= '1';
        idj_check1 <= '1';
        if(tpm_en = '1')then
            BP_to_tpm_cmd <= B_MAKE_TESLA_CMD;
            next_state <= make_tesla_state1;
        else
            BP_to_tpm_cmd <= INVALID_TPM_CMD;
            next_state <= fresh_k_i_state2;
        end if;
    when make_tesla_state1 =>
        BP_cpu_ready <= '0';
        BP_time_begin <= '1';
        BP_to_zm_en <= '0';
        BP_to_zm_cmd <= INVALID_ZM_CMD;
        BP_to_tpm_cmd <= INVALID_TPM_CMD;
        tesla_init_check <= '1';
        idj_check1 <= '1';
        if(io_ready = '1')then
            BP_to_con_cmd <= WRITE_I;
            next_state <= make_tesla_state2;
        else
            BP_to_con_cmd <= INVALID_CON_CMD;
            next_state <= make_tesla_state1;
        end if;
    when make_tesla_state2 =>
        BP_cpu_ready <= '0';
        BP_time_begin <= '1';
        BP_to_zm_en <= '0';
        BP_to_tpm_cmd <= INVALID_TPM_CMD;
        BP_to_zm_cmd <= INVALID_ZM_CMD;
        tesla_init_check <= '1';
        idj_check1 <= '1';
        if(io_ready = '1')then
            BP_to_con_cmd <= READ_DATA;
            next_state <= make_tesla_state3;
        else
            BP_to_con_cmd <= INVALID_CON_CMD;
            next_state <= make_tesla_state2;
        end if;
    when make_tesla_state3 =>
```

```
        BP_cpu_ready <= '0';
        BP_time_begin <= '1';
        BP_to_con_cmd <= INVALID_CON_CMD;
        tesla_init_check <= '1';
        idj_check1 <= '1';
        -- 通知 ZM 发送
        if(tpm_en = '1')then
            BP_to_zm_en <= '1';
            BP_to_zm_cmd <= SEND_TESLA_CMD;
            BP_to_tpm_cmd <= B_FRESH_K_T_I_CMD;
            next_state <= make_tesla_state4;
        else
            BP_to_zm_en <= '0';
            BP_to_zm_cmd <= INVALID_ZM_CMD;
            BP_to_tpm_cmd <= INVALID_TPM_CMD;
            next_state <= make_tesla_state3;
        end if;
when make_tesla_state4 =>
        BP_cpu_ready <= '0';
        BP_time_begin <= '1';
        BP_to_con_cmd <= INVALID_CON_CMD;
        BP_to_zm_en <= '0';
        BP_to_zm_cmd <= INVALID_ZM_CMD;
        BP_to_tpm_cmd <= INVALID_TPM_CMD;
        tesla_init_check <= '1';
        idj_check1 <= '1';
        --
        if(tpm_en = '1')then
            next_state <= wait_state;
        else
            next_state <= make_tesla_state4;
        end if;
--------------------------------------------------------------------------------
-- 基站接收数据，发送给电脑基站的处理
--------------------------------------------------------------------------------
when enc_dec_snd_state1 =>
        BP_cpu_ready <= '0';
        BP_time_begin <= '1';
        BP_to_zm_en <= '0';
        BP_to_zm_cmd <= INVALID_ZM_CMD;
        BP_to_con_cmd <= INVALID_CON_CMD;
        tesla_init_check <= '1';
        idj_check1 <= '1';
        -- 通知 TPM 解密数据 K_i
```

```
        if( tpm_en = '1' ) then
            BP_to_tpm_cmd <= B_DEC_K_I_CMD;
            next_state <= enc_dec_snd_state2;
        else
            BP_to_tpm_cmd <= INVALID_TPM_CMD;
            next_state <= enc_dec_snd_state1;
        end if;
    when enc_dec_snd_state2 =>
        BP_cpu_ready <= '0';
        BP_time_begin <= '1';
        BP_to_tpm_cmd <= INVALID_TPM_CMD;
        BP_to_zm_en <= '0';
        BP_to_zm_cmd <= INVALID_ZM_CMD;
        tesla_init_check <= '1';
        idj_check1 <= '1';
        -- 需要解密数据
        if( io_ready = '1') then
            BP_to_con_cmd <= WRITE_DATA;
            next_state <= hash_verify_state1;
        else
            BP_to_con_cmd <= INVALID_CON_CMD;
            next_state <= enc_dec_snd_state2;
        end if;
    when hash_verify_state1 =>
        BP_cpu_ready <= '0';
        BP_time_begin <= '1';
        BP_to_tpm_cmd <= INVALID_TPM_CMD;
        BP_to_zm_en <= '0';
        BP_to_zm_cmd <= INVALID_ZM_CMD;
        BP_to_con_cmd <= INVALID_CON_CMD;
        tesla_init_check <= '1';
        idj_check1 <= '1';
        -- Hash 验证正确,读数据进行下一步加密操作
        if (hash_result = HASH_SUC) then
            next_state <= enc_dec_snd_state3;
        elsif(hash_result = HASH_FAIL)then
            next_state <= hash_verify_state2;
        else
            next_state <= hash_verify_state1;
        end if;
    when hash_verify_state2 =>
        BP_cpu_ready <= '0';
        BP_time_begin <= '1';
        BP_to_tpm_cmd <= INVALID_TPM_CMD;
```

```
        BP_to_zm_en <= '0';
        BP_to_zm_cmd <= INVALID_ZM_CMD;
        tesla_init_check <= '1';
        idj_check1 <= '1';
        -- Hash 验证错误，把 TPM 的  获量
        if( io_ready = '1') then
            BP_to_con_cmd <= READ_INVALID_DATA_CMD;
            next_state <= hash_verify_state3;
        else
            BP_to_con_cmd <= INVALID_CON_CMD;
            next_state <= hash_verify_state2;
        end if;
    when hash_verify_state3 =>
        BP_cpu_ready <= '0';
        BP_time_begin <= '1';
        BP_to_tpm_cmd <= INVALID_TPM_CMD;
        BP_to_zm_en <= '0';
        BP_to_zm_cmd <= INVALID_ZM_CMD;
        BP_to_con_cmd <= INVALID_CON_CMD;
        tesla_init_check <= '1';
        idj_check1 <= '1';
        -- 数据读空后，转为等待状态
        if( tpm_en = '1') then
            next_state <= wait_state;
        else
            next_state <= hash_verify_state3;
        end if;
    -- 把解密结果送入 fifo2 --
    when enc_dec_snd_state3 =>
        BP_cpu_ready <= '0';
        BP_time_begin <= '1';
        BP_to_tpm_cmd <= INVALID_TPM_CMD;
        BP_to_zm_en <= '0';
        BP_to_zm_cmd <= INVALID_ZM_CMD;
        tesla_init_check <= '1';
        idj_check1 <= '1';
        -- 解密完成,传送数据 入 fifo2
        if( io_ready = '1') then
            BP_to_con_cmd <= READ_DATA_TO_FIFO2;
            next_state <= enc_dec_snd_state4;
        else
            BP_to_con_cmd <= INVALID_CON_CMD;
            next_state <= enc_dec_snd_state3;
        end if;
```

```
when enc_dec_snd_state4 =>
        BP_cpu_ready <= '0';
        BP_time_begin <= '1';
        BP_to_tpm_cmd <= INVALID_TPM_CMD;
        BP_to_zm_en <= '0';
        BP_to_zm_cmd <= INVALID_ZM_CMD;
        BP_to_con_cmd <= INVALID_CON_CMD;
        tesla_init_check <= '1';
        idj_check1 <= '1';
        --   数据传送完成, 开始加密  ---
        if( tpm_en = '1') then
            BP_to_tpm_cmd <= B_ENC_K_P_CMD;
            next_state <= enc_dec_snd_state5;
        else
            BP_to_tpm_cmd <= INVALID_TPM_CMD;
            next_state <= enc_dec_snd_state4;
        end if;
when enc_dec_snd_state5 =>
        BP_cpu_ready <= '0';
        BP_time_begin <= '1';
        BP_to_tpm_cmd <= INVALID_TPM_CMD;
        BP_to_zm_en <= '0';
        BP_to_zm_cmd <= INVALID_ZM_CMD;
        tesla_init_check <= '1';
        idj_check1 <= '1';
        -- 开始加密,需要数据
        if( io_ready = '1') then
            BP_to_con_cmd <= WRITE_DATA_FROM_FIFO2;
            next_state <= enc_dec_snd_state6;
        else
            BP_to_con_cmd <= INVALID_CON_CMD;
            next_state <= enc_dec_snd_state5;
        end if;
when enc_dec_snd_state6 =>
        BP_cpu_ready <= '0';
        BP_time_begin <= '1';
        BP_to_tpm_cmd <= INVALID_TPM_CMD;
        BP_to_zm_en <= '0';
        BP_to_zm_cmd <= INVALID_ZM_CMD;
        tesla_init_check <= '1';
        idj_check1 <= '1';
        -- 加密完成，数据输出
        if( io_ready = '1') then
            BP_to_con_cmd <= READ_DATA;
```

```
                next_state <= enc_dec_snd_state7;
            else
                BP_to_con_cmd <= INVALID_CON_CMD;
                next_state <= enc_dec_snd_state6;
            end if;
        when enc_dec_snd_state7 =>
            BP_cpu_ready <= '0';
            BP_time_begin <= '1';
            BP_to_tpm_cmd <= INVALID_TPM_CMD;
            BP_to_con_cmd <= INVALID_CON_CMD;
            tesla_init_check <= '1';
            idj_check1 <= '1';
            -- 数据输出完成，通知 ZM 发送
            if( tpm_en = '1') then
                BP_to_zm_en <= '1';
                BP_to_zm_cmd <= SEND_TO_COMPUTER;
                next_state <= wait_state;
            else
                BP_to_zm_en <= '0';
                BP_to_zm_cmd <= INVALID_ZM_CMD;
                next_state <= enc_dec_snd_state7;
            end if;
        end case;
else
        -- 系统复位或侵毓贵?en  = 0 ，那么  鲋低辰  氤跏化状? ---
        tesla_init_check <= '0';
        idj_check1 <= '0';
        BP_to_zm_en <= '0';
        BP_to_con_cmd <= INVALID_CON_CMD;
        BP_to_zm_cmd <= INVALID_ZM_CMD;
        BP_to_tpm_cmd <= INVALID_TPM_CMD;
        BP_cpu_ready <= '1';
        BP_time_begin <= '0';
        next_state <= init_state1;
end if;
end process;

end RTL;
```

--- cpu_base_counter.vhd

```
entity cpu_base_counter is
        port(clk        :       IN          STD_LOGIC;
                rst         :       in          std_logic;
```

```
                en              :   in          std_logic;
                time_end :      out  std_logic
                );
end cpu_base_counter;

architecture RTL of cpu_base_counter is

signal BP_time_end      : std_logic;
signal counter :    std_logic_vector(COUNTER_WIDTH-1 DOWNTO 0);
begin
    process(clk, rst)
    begin

        if(rst = '1')then
        counter <= DISPATCH_TIME;
            BP_time_end <= '0';
        elsif clk'event and clk = '1' then
            if(en = '1')then
                if(counter = DEF_ZERO)then
                    BP_time_end <= '1';
                    counter <= DISPATCH_TIME;
                else
                    BP_time_end <= '0';
                    counter <= counter - 1;
                end if;
            else
                counter <= DISPATCH_TIME;
            end if;
        end if;
    end process;

    time_end <= BP_time_end;
end RTL;
```

--- cpu_client.vhd

```
entity cpu_client is
    Port ( clk : in    STD_LOGIC;
           rst : in    STD_LOGIC;
           c_en : in    STD_LOGIC;
             tpm_en : in std_logic;
             io_ready : in std_logic;
           verify_result : in    STD_LOGIC_VECTOR (VERIFY_WIDTH-1 downto 0);
```

```
                oneway_finish : in    STD_LOGIC;
                from_zm_en : in    STD_LOGIC;
                from_zm_cmd : in    STD_LOGIC_VECTOR (ZM_CMD_WIDTH-1 downto 0);
                to_zm_en : out    STD_LOGIC;
                to_zm_cmd : out    STD_LOGIC_VECTOR (ZM_CMD_WIDTH-1 downto 0);
                to_tpm_cmd : out    STD_LOGIC_VECTOR (TPM_CMD_WIDTH-1 downto 0);
                to_con_cmd : out    STD_LOGIC_VECTOR (CONTROL_CMD_WIDTH-1 downto
0);
                zm_ready : in    STD_LOGIC;
                con_ready : in    STD_LOGIC;
                cpu_ready : out    STD_LOGIC
                );
end cpu_client;

architecture RTL of cpu_client is

type state is(init_state1, init_state2, init_state3,
                        dec_tesla_init_state1, dec_tesla_init_state2, dec_tesla_init_state3,
                        dec_tesla_init_state4, dec_tesla_init_state5,
                        wait_state,
                        dec_tesla_state1, dec_tesla_state2, dec_tesla_state3,
                        dec_tesla_state4, dec_tesla_state5, dec_tesla_state6,
                        query_verify_state1,
                        oneway_state1, oneway_state2, oneway_state3, oneway_state4,
                        erro_handle_state1,
                        verify_k_ti_state1,        verify_k_ti_state2,        verify_k_ti_state3,
verify_k_ti_state4, verify_k_ti_state5,
                        enc_snd_state1, enc_snd_state2, enc_snd_state3, enc_snd_state4,
                        hash_verify_state1, hash_verify_state2, hash_verify_state3,
                        hash_verify_state21, hash_verify_state22, hash_verify_state23);
signal curr_state, next_state : state;
signal BP_to_zm_en : std_logic;
signal BP_to_con_cmd : STD_LOGIC_VECTOR (CONTROL_CMD_WIDTH-1 downto 0);
signal BP_to_zm_cmd : STD_LOGIC_VECTOR (ZM_CMD_WIDTH-1 downto 0);
signal BP_to_tpm_cmd : STD_LOGIC_VECTOR (TPM_CMD_WIDTH-1 downto 0);
signal BP_cpu_ready : std_logic;
begin

PROCESS(clk, rst)
begin
    if rst = '1' then
            curr_state <= init_state1;
        elsif clk'event and clk='1' then
            curr_state <= next_state;
```

```
        end if;

end process;
```

```
to_tpm_cmd <= BP_to_tpm_cmd;
                to_con_cmd <= BP_to_con_cmd;
                to_zm_en <= BP_to_zm_en;
                to_zm_cmd <= BP_to_zm_cmd;
                cpu_ready <= BP_cpu_ready;

process(curr_state, c_en,  verify_result, from_zm_en, from_zm_cmd, zm_ready, con_ready, io_ready, tpm_en,
                oneway_finish )


begin
if( c_en = '1' ) then

    case curr_state is
    --------------------------------------------------------------------------------
    -- ID 基的处理
    --------------------------------------------------------------------------------
    when init_state1 =>
        BP_cpu_ready <= '0';
        BP_to_tpm_cmd <= INVALID_TPM_CMD;
        BP_to_zm_en <= '0';
        BP_to_zm_cmd <= INVALID_ZM_CMD;

        if(io_ready = '1')then
            -- ?data_control 把  ID 基 送入 TPM
            BP_to_con_cmd <= WRITE_DATA;
            next_state <= init_state2;
        else
            BP_to_con_cmd <= INVALID_CON_CMD;
            next_state <= init_state1;
        end if;
    when init_state2 =>
        BP_cpu_ready <= '0';
        BP_to_tpm_cmd <= INVALID_TPM_CMD;
        BP_to_zm_en <= '0';
        BP_to_con_cmd <= INVALID_CON_CMD;
        BP_to_zm_cmd <= INVALID_ZM_CMD;
```

```
        -- TPM 完成了 ID 基 的处理
        if(tpm_en = '1') then
            next_state <= wait_state;
        else
            next_state <= init_state2;
        end if;
-------------------------------------------------------------------------------
-- 节点接收到 TESlA 初始包的处理
-------------------------------------------------------------------------------
when init_state3 =>
        BP_cpu_ready <= '0';
        BP_to_zm_en <= '0';
        BP_to_con_cmd <= INVALID_CON_CMD;
        BP_to_zm_cmd <= INVALID_ZM_CMD;

        if(tpm_en = '1') then
            --  通知 TPM 用 K 子 解密 TESLA 初始包
            BP_to_tpm_cmd <= C_DEC_TESLA_INIT_CMD;
            next_state <= dec_tesla_init_state1;
        else
            BP_to_tpm_cmd <= INVALID_TPM_CMD;
            next_state <= init_state3;
        end if;
-------------------------------------------------------------------------------
-- 节点和 TPM 之间   TESLA 初始包的交互
-------------------------------------------------------------------------------
when dec_tesla_init_state1 =>
        BP_cpu_ready <= '0';
        BP_to_tpm_cmd <= INVALID_TPM_CMD;
        BP_to_zm_en <= '0';
        BP_to_zm_cmd <= INVALID_ZM_CMD;

        if( io_ready = '1') then
            BP_to_con_cmd <= WRITE_DATA;
            next_state <= hash_verify_state1;
        else
            BP_to_con_cmd <= INVALID_CON_CMD;
            next_state <= dec_tesla_init_state1;
        end if;
-- 验证 TESLA 初始包 是否正确
when hash_verify_state1 =>
        BP_to_tpm_cmd <= INVALID_TPM_CMD;
        BP_to_zm_en <= '0';
        BP_to_con_cmd <= INVALID_CON_CMD;
```

```
        BP_to_zm_cmd <= INVALID_ZM_CMD;

        BP_cpu_ready <='0';
        -- Hash 验证正确,读数据进行下一步加密操作
        if (verify_result = HASH_SUC) then
            next_state <= dec_tesla_init_state2;
        elsif(verify_result = HASH_FAIL)then
            next_state <= hash_verify_state2;
        else
            next_state <= hash_verify_state1;
        end if;
    when hash_verify_state2 =>
        BP_to_tpm_cmd <= INVALID_TPM_CMD;
        BP_to_zm_en <= '0';
        BP_to_zm_cmd <= INVALID_ZM_CMD;

        BP_cpu_ready <='0';
        -- Hash 验证错误，把 TPM 的数据?
        if( io_ready = '1') then
            BP_to_con_cmd <= READ_INVALID_DATA_CMD;
            next_state <= hash_verify_state3;
        else
            BP_to_con_cmd <= INVALID_CON_CMD;
            next_state <= hash_verify_state2;
        end if;
    when hash_verify_state3 =>
        BP_to_tpm_cmd <= INVALID_TPM_CMD;
        BP_to_zm_en <= '0';
        BP_to_con_cmd <= INVALID_CON_CMD;
        BP_to_zm_cmd <= INVALID_ZM_CMD;

        BP_cpu_ready <='0';
        -- 数据读空，跳转到空闲态
        if( tpm_en = '1') then
            next_state <= wait_state;
        else
            next_state <= hash_verify_state3;
        end if;
    -- HASH 成功后处理
    when dec_tesla_init_state2 =>
        BP_cpu_ready <= '0';
        BP_to_tpm_cmd <= INVALID_TPM_CMD;
        BP_to_zm_en <= '0';
        BP_to_zm_cmd <= INVALID_ZM_CMD;
```

```
        if( io_ready = '1') then
            BP_to_con_cmd <= READ_TESLA_INIT;
            next_state <= dec_tesla_init_state3;
        else
            BP_to_con_cmd <= INVALID_CON_CMD;
            next_state <= dec_tesla_init_state2;
        end if;
    when dec_tesla_init_state3 =>
        BP_cpu_ready <= '0';
        BP_to_zm_en <= '0';
        BP_to_con_cmd <= INVALID_CON_CMD;
        BP_to_zm_cmd <= INVALID_ZM_CMD;

        if( tpm_en = '1') then
            -- 通知 TPM 更新 K_To, Ko
            BP_to_tpm_cmd <= C_REFRESH_TESLA_INIT;
            next_state <= dec_tesla_init_state4;
        else
            BP_to_tpm_cmd <= INVALID_TPM_CMD;
            next_state <= dec_tesla_init_state3;
        end if;
    when dec_tesla_init_state4 =>
        BP_cpu_ready <= '0';
        BP_to_tpm_cmd <= INVALID_TPM_CMD;
        BP_to_zm_en <= '0';
        BP_to_zm_cmd <= INVALID_ZM_CMD;

        if( io_ready = '1') then
            --  通知 control ? K_To, Ko 发送给 TPM
            BP_to_con_cmd <= WRITE_TESLA_INIT;
            next_state <= dec_tesla_init_state5;
        else
            BP_to_con_cmd <= INVALID_CON_CMD;
            next_state <= dec_tesla_init_state4;
        end if;
    when dec_tesla_init_state5 =>
        BP_cpu_ready <= '0';
        BP_to_tpm_cmd <= INVALID_TPM_CMD;
        BP_to_zm_en <= '0';
        BP_to_con_cmd <= INVALID_CON_CMD;
        BP_to_zm_cmd <= INVALID_ZM_CMD;

        if( tpm_en = '1') then
```

```
            --  更新 K_To, Ko 完成
            next_state <= wait_state;
        else
            next_state <= dec_tesla_init_state5;
        end if;
-------------------------------------------------------------------------------
-- 等待状态
-------------------------------------------------------------------------------
when wait_state =>
        BP_to_zm_en <= '0';
        BP_to_tpm_cmd <= INVALID_TPM_CMD;
        BP_to_con_cmd <= INVALID_CON_CMD;
        BP_to_zm_cmd <= INVALID_zM_CMD;

        -- 收到 TESLA 初始包
        -- 有数据要发送（加密，hash）
        -- 收到 TESLA 密钥包
        if(from_zm_en = '1') then
            BP_cpu_ready <= '0';
            case from_zm_cmd is
            when REV_TESLA_INIT_CMD => next_state <= init_state3;
            when ENC_SND_DATA_CMD => next_state <= enc_snd_state1;
            when REV_TESLA_CMD => next_state <= dec_tesla_state1;
            when others => next_state <= wait_state;
            end case;
        else
            BP_cpu_ready <= '1';
            next_state <= wait_state;
        end if;
-------------------------------------------------------------------------------
-- 发送从 PS2 口接受的数据
-------------------------------------------------------------------------------
when enc_snd_state1     =>
        BP_cpu_ready <= '0';
        BP_to_zm_en <= '0';
        BP_to_con_cmd <= INVALID_CON_CMD;
        BP_to_zm_cmd <= INVALID_ZM_CMD;

        if( tpm_en = '1') then
            -- 通知 TPM 加密从 ps2 口接受的数据, 用 Ki
            BP_to_tpm_cmd <= C_ENC_K_I_CMD;
            next_state <= enc_snd_state2;
        else
            BP_to_tpm_cmd <= INVALID_TPM_CMD;
```

```
            next_state <= enc_snd_state1;
        end if;
    when enc_snd_state2     =>
        BP_cpu_ready <= '0';
        BP_to_tpm_cmd <= INVALID_TPM_CMD;
        BP_to_zm_en <= '0';
        BP_to_zm_cmd <= INVALID_ZM_CMD;

        if( io_ready = '1') then
            -- 通知 controll 传送数据
            BP_to_con_cmd <= WRITE_DATA;
            next_state <= enc_snd_state3;
        else
            BP_to_con_cmd <= INVALID_CON_CMD;
            next_state <= enc_snd_state2;
        end if;
    when enc_snd_state3     =>
        BP_cpu_ready <= '0';
        BP_to_tpm_cmd <= INVALID_TPM_CMD;
        BP_to_zm_en <= '0';
        BP_to_zm_cmd <= INVALID_ZM_CMD;

        if( io_ready = '1') then
            -- 通知 control 传送 加密后的数据
            BP_to_con_cmd <= READ_DATA;
            next_state <= enc_snd_state4;
        else
            BP_to_con_cmd <= INVALID_CON_CMD;
            next_state <= enc_snd_state3;
        end if;
    when enc_snd_state4     =>
        BP_cpu_ready <= '0';
        BP_to_tpm_cmd <= INVALID_TPM_CMD;
        BP_to_con_cmd <= INVALID_CON_CMD;

        if( tpm_en = '1' ) then
            -- 通知 ZM 发送加密后的数据
            BP_to_zm_en <= '1';
            BP_to_zm_cmd <= SND_DATA_CMD;
            next_state <= wait_state;
        else
            BP_to_zm_en <= '0';
            BP_to_zm_cmd <= INVALID_ZM_CMD;
            next_state <= enc_snd_state4;
```

```
        end if;
-------------------------------------------------------------------------
-- 收到 TESLA 密钥包
-------------------------------------------------------------------------
when dec_tesla_state1 =>
        BP_cpu_ready <= '0';
        BP_to_zm_en <= '0';
        BP_to_con_cmd <= INVALID_CON_CMD;
        BP_to_zm_cmd <= INVALID_ZM_CMD;

        if( tpm_en = '1') then
            -- 通知 TPM 开始 HAsh TESLA 包
            BP_to_tpm_cmd <= C_HASN_CMD;
            next_state <= dec_tesla_state2;
        else
            BP_to_tpm_cmd <= INVALID_TPM_CMD;
            next_state <= dec_tesla_state1;
        end if;
when dec_tesla_state2 =>
        BP_cpu_ready <= '0';
        BP_to_tpm_cmd <= INVALID_TPM_CMD;
        BP_to_zm_en <= '0';
        BP_to_zm_cmd <= INVALID_ZM_CMD;

        if( io_ready = '1') then
            -- TPM 需要数据
            BP_to_con_cmd <= WRITE_DATA;
            next_state <= hash_verify_state21;
        else
            BP_to_con_cmd <= INVALID_CON_CMD;
            next_state <= dec_tesla_state2;
        end if;
when hash_verify_state21 =>
        BP_to_tpm_cmd <= INVALID_TPM_CMD;
        BP_to_zm_en <= '0';
        BP_to_con_cmd <= INVALID_CON_CMD;
        BP_to_zm_cmd <= INVALID_ZM_CMD;

        BP_cpu_ready <='0';
        -- Hash 验证正确,读数据进行下一步加密操作
        if (verify_result = HASH_SUC) then
            next_state <= dec_tesla_state3;
        elsif(verify_result = HASH_FAIL)then
            next_state <= hash_verify_state22;
```

```
        else
            next_state <= hash_verify_state21;
        end if;
    when hash_verify_state22 =>
        BP_to_tpm_cmd <= INVALID_TPM_CMD;
        BP_to_zm_en <= '0';
        BP_to_zm_cmd <= INVALID_ZM_CMD;

        BP_cpu_ready <='0';
        -- Hash 验证错误，把 TPM 的数据读空
        if( io_ready = '1') then
            BP_to_con_cmd <= READ_INVALID_DATA_CMD;
            next_state <= hash_verify_state23;
        else
            BP_to_con_cmd <= INVALID_CON_CMD;
            next_state <= hash_verify_state22;
        end if;
    when hash_verify_state23 =>
        BP_to_tpm_cmd <= INVALID_TPM_CMD;
        BP_to_zm_en <= '0';
        BP_to_con_cmd <= INVALID_CON_CMD;
        BP_to_zm_cmd <= INVALID_ZM_CMD;

        BP_cpu_ready <='0';
        -- 数据读空后，转为等待状态
        if( tpm_en = '1') then
            next_state <= wait_state;
        else
            next_state <= hash_verify_state23;
        end if;
    -- HASH 验证正确， 先读取数据
    when dec_tesla_state3 =>
        BP_cpu_ready <= '0';
        BP_to_tpm_cmd <= INVALID_TPM_CMD;
        BP_to_zm_en <= '0';
        BP_to_zm_cmd <= INVALID_ZM_CMD;

        if( io_ready = '1') then
            -- 把 TESLA 包读出? ----
            BP_to_con_cmd <= READ_TESLA;
            next_state <= dec_tesla_state4;
        else
            BP_to_con_cmd <= INVALID_CON_CMD;
            next_state <= dec_tesla_state3;
```

```
        end if;
    when dec_tesla_state4 =>
        BP_cpu_ready <= '0';
        BP_to_tpm_cmd <= INVALID_TPM_CMD;
        BP_to_zm_en <= '0';
        BP_to_con_cmd <= INVALID_CON_CMD;
        BP_to_zm_cmd <= INVALID_ZM_CMD;

        if( tpm_en = '1') then
            -- 把 TESLA 包读出去后，查询是否需要单向
            next_state <= query_verify_state1;
        else
            next_state <= dec_tesla_state4;
        end if;
    when query_verify_state1 =>
        BP_cpu_ready <= '0';
        BP_to_tpm_cmd <= INVALID_TPM_CMD;
        BP_to_zm_en <= '0';
        BP_to_con_cmd <= INVALID_CON_CMD;
        BP_to_zm_cmd <= INVALID_ZM_CMD;

        -- 查询 data_control 里面的 i ---
        if (oneway_finish = '1') then
            next_state <= verify_k_ti_state1;
        else
            next_state <= oneway_state1;
        end if;
    ------------------------------------------------------------------------------
    --  继续对 K_Ti 进行单向函数
    ------------------------------------------------------------------------------
    when oneway_state1 =>
        BP_cpu_ready <= '0';
        BP_to_zm_en <= '0';
        BP_to_con_cmd <= INVALID_CON_CMD;
        BP_to_zm_cmd <= INVALID_ZM_CMD;

        -- 通知 TPM 开始 单向
        if (tpm_en = '1') then
            BP_to_tpm_cmd <= C_ONEWAY_K_T_I_CMD;
            next_state <= oneway_state2;
        else
            BP_to_tpm_cmd <= INVALID_TPM_CMD;
            next_state <= oneway_state1;
        end if;
```

```
when oneway_state2 =>
        BP_cpu_ready <= '0';
        BP_to_tpm_cmd <= INVALID_TPM_CMD;
        BP_to_zm_en <= '0';
        BP_to_con_cmd <= INVALID_CON_CMD;
        BP_to_zm_cmd <= INVALID_ZM_CMD;

        -- TPM 单向 需要数据
        if (io_ready = '1') then
            BP_to_con_cmd <= WRITE_K_T;
            next_state <= oneway_state3;
        else
            BP_to_con_cmd <= INVALID_CON_CMD;
            next_state <= oneway_state2;
        end if;
when oneway_state3 =>
        BP_cpu_ready <= '0';
        BP_to_tpm_cmd <= INVALID_TPM_CMD;
        BP_to_zm_en <= '0';
        BP_to_zm_cmd <= INVALID_ZM_CMD;

        -- 查询 data_control 里面的 i 的差值
        if (io_ready = '1') then
            BP_to_con_cmd <= READ_K_T;
            next_state <= oneway_state4;
        else
            BP_to_con_cmd <= INVALID_CON_CMD;
            next_state <= oneway_state3;
        end if;
when oneway_state4 =>
        BP_cpu_ready <= '0';
        BP_to_tpm_cmd <= INVALID_TPM_CMD;
        BP_to_zm_en <= '0';
        BP_to_con_cmd <= INVALID_CON_CMD;
        BP_to_zm_cmd <= INVALID_ZM_CMD;

        -- 查询 data_control 里面的 i 的差值
        if (tpm_en = '1') then
            next_state <= query_verify_state1;
        else
            next_state <= oneway_state4;
        end if;
--------------------------------------------------------------------------------
-- 验证 K_Ti 的状态
```

```
-------------------------------------------------------------------------------
when verify_k_ti_state1 =>
        BP_cpu_ready <= '0';
        BP_to_zm_en <= '0';
        BP_to_con_cmd <= INVALID_CON_CMD;
        BP_to_zm_cmd <= INVALID_ZM_CMD;

        -- 通知 TPM 验证 K_Ti
        if (tpm_en = '1') then
            BP_to_tpm_cmd <= C_VERIFY_K_T_I_CMD;
            next_state <= verify_k_ti_state2;
        else
            BP_to_tpm_cmd <= INVALID_TPM_CMD;
            next_state <= verify_k_ti_state1;
        end if;
when verify_k_ti_state2 =>
        BP_cpu_ready <= '0';
        BP_to_tpm_cmd <= INVALID_TPM_CMD;
        BP_to_zm_en <= '0';
        BP_to_zm_cmd <= INVALID_ZM_CMD;

        --  TPM 验证 K_Ti 需要数据
        if (io_ready = '1') then
            BP_to_con_cmd <= WRITE_K_T;
            next_state <= verify_k_ti_state3;
        else
            BP_to_con_cmd <= INVALID_CON_CMD;
            next_state <= verify_k_ti_state2;
        end if;
when verify_k_ti_state3 =>
        BP_cpu_ready <= '0';
        BP_to_tpm_cmd <= INVALID_TPM_CMD;
        BP_to_zm_en <= '0';
        BP_to_con_cmd <= INVALID_CON_CMD;
        BP_to_zm_cmd <= INVALID_ZM_CMD;

        -- 验证结果
        if (verify_result = VERIFY_SUC) then
            next_state <= verify_k_ti_state4;
        elsif(verify_result = VERIFY_FAIL) then
            next_state <= erro_handle_state1;
        else
            next_state <= verify_k_ti_state3;
        end if;
```

```
-- 验证结果错误
when erro_handle_state1 =>
        BP_cpu_ready <= '0';
        BP_to_tpm_cmd <= INVALID_TPM_CMD;
        BP_to_zm_en <= '0';
        BP_to_con_cmd <= INVALID_CON_CMD;
        BP_to_zm_cmd <= INVALID_ZM_CMD;

        if (tpm_en = '1') then
            next_state <= wait_state;
        else
            next_state <= erro_handle_state1;
        end if;
-- 验证结果正确
when verify_k_ti_state4 =>
        BP_cpu_ready <= '0';
        BP_to_tpm_cmd <= INVALID_TPM_CMD;
        BP_to_zm_en <= '0';
        BP_to_zm_cmd <= INVALID_ZM_CMD;

        -- 正确需要更新 K_Ti
        if (io_ready = '1') then
            BP_to_con_cmd <= WRTIT_NEW_K_T;
            next_state <= verify_k_ti_state5;
        else
            BP_to_con_cmd <= INVALID_CON_CMD;
            next_state <= verify_k_ti_state4;
        end if;
when verify_k_ti_state5 =>
        BP_cpu_ready <= '0';
        BP_to_zm_en <= '0';
        BP_to_con_cmd <= INVALID_CON_CMD;
        BP_to_zm_cmd <= INVALID_ZM_CMD;

        -- 更新完成，通知  TPM 开始解密 E(())
        if (tpm_en = '1') then
            BP_to_tpm_cmd <= C_DEC_TESLA_CMD;
            next_state <= dec_tesla_state5;
        else
            BP_to_tpm_cmd <= INVALID_TPM_CMD;
            next_state <= verify_k_ti_state5;
        end if;
--------------------------------------------------------------------------------
-- 解密 E(      (     )      )  状态
```

```
    ---------------------------------------------------------------------------
    when dec_tesla_state5 =>
        BP_cpu_ready <= '0';
        BP_to_tpm_cmd <= INVALID_TPM_CMD;
        BP_to_zm_en <= '0';
        BP_to_zm_cmd <= INVALID_ZM_CMD;

        -- 给 TPM 传送  E(     (    )     )
        if (io_ready = '1') then
            BP_to_con_cmd <= WRITE_E_K_I;
            next_state <= dec_tesla_state6;
        else
            BP_to_con_cmd <= INVALID_CON_CMD;
            next_state <= dec_tesla_state5;
        end if;
    when dec_tesla_state6 =>
        BP_cpu_ready <= '0';
        BP_to_tpm_cmd <= INVALID_TPM_CMD;
        BP_to_zm_en <= '0';
        BP_to_con_cmd <= INVALID_CON_CMD;
        BP_to_zm_cmd <= INVALID_ZM_CMD;

        -- 解密完成， TESLA 更新操作默认在 data_control 里面做
        if (tpm_en = '1') then
            next_state <= wait_state;
        else
            next_state <= dec_tesla_state6;
        end if;
    end case;

else
    -- 系掣次缓?en = 0 ，那么整个系统进入初始化状态
    BP_cpu_ready <= '1';
    BP_to_zm_en <= '0';
    BP_to_con_cmd <= INVALID_CON_CMD;
    BP_to_zm_cmd <= INVALID_ZM_CMD;
    BP_to_tpm_cmd <= INVALID_TPM_CMD;
    BP_cpu_ready <= '1';

    next_state <= init_state1;
end if;

end process;
end RTL;
```

-- data_control_pack.vhd

```
package data_control_pack is

   -- 命令的宽度
   constant CONTROL_CMD_WIDTH       : INTEGER := 4;


   -- INVALID CMD
   CONSTANT INVALID_CON_CMD:  STD_LOGIC_VECTOR(CONTROL_CMD_WIDTH-1
downto 0) := "0000";
   -- 从 TPM 读正常的数据 写入 FIFO
   constant READ_DATA          :        STD_LOGIC_VECTOR(CONTROL_CMD_WIDTH-1
downto 0) := "0001";
   -- 将数据从 FIFO 写入 TPM
   constant WRITE_DATA   :  STD_LOGIC_VECTOR(CONTROL_CMD_WIDTH-1   downto
0) := "0010";

   -- 把 TPM 读入 FIFO2，主要暂存解密完成的数据
   CONSTANT                          READ_DATA_TO_FIFO2                    :
STD_LOGIC_VECTOR(CONTROL_CMD_WIDTH-1 downto 0) := "0011";
 -- 从 fifo2 中把数据读入 TPM
 CONSTANT                        WRITE_DATA_FROM_FIFO2                   :
STD_LOGIC_VECTOR(CONTROL_CMD_WIDTH-1 downto 0) := "0100";

    --  从 TPM 读解密后的 TESLA 初始包
   constant READ_TESLA_INIT       :  STD_LOGIC_VECTOR(CONTROL_CMD_WIDTH-1
downto 0) := "0101";
   --  把从 TPM 读取的解密后的 TESLA 初始包的内容分开后送回?TPM
   constant WRITE_TESLA_INIT      :  STD_LOGIC_VECTOR(CONTROL_CMD_WIDTH-1
downto 0) := "0110";

   -- 从 TPM 读 解密后的 TESLA 密钥包
   constant READ_TESLA   :  STD_LOGIC_VECTOR(CONTROL_CMD_WIDTH-1   downto
0) := "0111";
   -- 组建 TESLA 密钥包需要 i 值
   constant WRITE_I          :  STD_LOGIC_VECTOR(CONTROL_CMD_WIDTH-1   downto
0) := "1000";

   -- 把 K<Ti-1> 送入 TPM
   constant WRITE_K_T          :        STD_LOGIC_VECTOR(CONTROL_CMD_WIDTH-1
downto 0) := "1001";
   -- 从 TPM 读做了 单向函数的 K<Ti-1>
```

```
   constant READ_K_T        :   STD_LOGIC_VECTOR(CONTROL_CMD_WIDTH-1   downto
0) := "1010";

   -- 把櫓ふ  泛蟮?K_Ti 送给 TPM
   constant WRTIT_NEW_K_T : STD_LOGIC_VECTOR(CONTROL_CMD_WIDTH-1 downto
0) := "1011";

   -- 把 E（( )）送入 TPM
   constant WRITE_E_K_I   :   STD_LOGIC_VECTOR(CONTROL_CMD_WIDTH-1   downto
0):= "1100";

 -- 把 TPM 里面的无效数据读空
 CONSTANT READ_INVALID_DATA_CMD          :
STD_LOGIC_VECTOR(CONTROL_CMD_WIDTH-1 downto 0) := "1101";
 -- 往 TPM 中写空的数据
 constant WRITE_NULL_CMD                 :
STD_LOGIC_VECTOR(CONTROL_CMD_WIDTH-1 downto 0) := "1110";

   constant DATA_WIDTH    : INTEGER := 32;

   -- TESLA_MEM 密钥包各内容

   -- 0 => 备用，下面往后退一位
   -- 0 => K_T_i   中间结果
   -- 1 => E(E())          i
   -- 2 => K<Ti-1>              i
   -- 3 => i 值             i   初始值为零，下面接收到的第一个 i 值也为 0
   -- 4 => E(E())         i+1
   -- 5 => K<Ti-1>       i+1
   -- 6 => i 值           i+1
   -- 7 => i 的差值
   -- 8 => 组建 TESLA 包需要 i
   -- 9 => K<T_0>    1001
   -- 10 => K_0       1010
   constant ADDR_WIDTH : integer := 4;
   TYPE MEM IS ARRAY(0 to 11) OF
        STD_LOGIC_VECTOR( DATA_WIDTH - 1 DOWNTO 0);

   constant GAOZU : STD_LOGIC_VECTOR(DATA_WIDTH-1 DOWNTO 0) := X"00000001";
end data_control_pack;
```

--top_data_control_top.vhd

```
entity top_data_control is
    Port (
            clk : in    STD_LOGIC;
            rst     : in    STD_LOGIC;
            reconfig : in std_logic;
            cpu_cmd  :  in    STD_LOGIC_VECTOR(CONTROL_CMD_WIDTH-1  downto  0);
                -- 命令内容
            tpm_empty : in    STD_LOGIC;
    -- TPM 空信号
            fifo_empty : in    STD_LOGIC;
    -- FIFO 空信号
            re_tpm : out    STD_LOGIC;
--  从 TPM 读
            wr_tpm : out    STD_LOGIC;
    --  写入 TPM
            re_fifo : out    STD_LOGIC;                                          --
从 fifo  读
            wr_fifo : out    STD_LOGIC;                                         --
写入  fifo
            io_finish : out    STD_LOGIC;                                       --
IO  完成
            verify_finish : out STD_LOGIC;                                      --
单向函数完成(验证  K<Ti-1>  完成)
            ready : out    STD_LOGIC;
    --  空闲信号
                tpm_data_in : IN STD_LOGIC_VECTOR(DATA_WIDTH-1 DOWNTO 0);
                tpm_data_out : out STD_LOGIC_VECTOR(DATA_WIDTH-1 DOWNTO 0);
                zm_data_in : in STD_LOGIC_VECTOR(DATA_WIDTH-1 DOWNTO 0);
                zm_data_out : out STD_LOGIC_VECTOR(DATA_WIDTH-1 DOWNTO 0)
                );
end top_data_control;

architecture RTL of top_data_control is

    component fifo_2
        port (
        clk: IN std_logic;
        din: IN std_logic_VECTOR(31 downto 0);
        rd_en: IN std_logic;
        srst: IN std_logic;
        wr_en: IN std_logic;
        dout: OUT std_logic_VECTOR(31 downto 0);
        empty: OUT std_logic;
        full: OUT std_logic
```

```
        );
end component;


COMPONENT data_control_top
PORT(
        clk : IN std_logic;
        rst : IN std_logic;
        reconfig : IN std_logic;
        cpu_cmd : IN std_logic_vector(3 downto 0);
        tpm_empty : IN std_logic;
        fifo_empty : IN std_logic;
        fifo2_empty : IN std_logic;
        re_tpm : OUT std_logic;
        wr_tpm : OUT std_logic;
        re_fifo : OUT std_logic;
        wr_fifo : OUT std_logic;
        re_fifo2 : OUT std_logic;
        wr_fifo2 : OUT std_logic;
        wr_mem : OUT std_logic;
        mem_addr : OUT std_logic_vector(3 downto 0);
        fresh_cmd : OUT std_logic_vector(1 downto 0);
        fresh_k_ti_i : OUT std_logic_vector(1 downto 0);
        IO_finish : OUT std_logic;
        ready : OUT std_logic;
        sel_mux : OUT std_logic_vector(1 downto 0)
        );
END COMPONENT;

COMPONENT tesla_mem
PORT(
        clk : IN std_logic;
        rst : IN std_logic;
        reconfig : IN std_logic;
        wr : IN std_logic;
        addr : IN std_logic_vector(3 downto 0);
        data_in : IN std_logic_vector(31 downto 0);
        fresh_cmd : IN std_logic_vector(1 downto 0);
        fresh_k_ti_i : IN std_logic_vector(1 downto 0);
        data_out : OUT std_logic_vector(31 downto 0);
        oneway_finish : OUT std_logic
        );
END COMPONENT;
```

```
    COMPONENT mux3_1by2sel
    PORT(
        a : IN std_logic_vector(DATA_WIDTH-1 downto 0);
        b : IN std_logic_vector(DATA_WIDTH-1 downto 0);
        c : IN std_logic_vector(DATA_WIDTH-1 downto 0);
        d : IN std_logic_vector(DATA_WIDTH-1 downto 0);
        se : IN std_logic_vector(1 downto 0);
        o : OUT std_logic_vector(DATA_WIDTH-1 downto 0)
        );
    END COMPONENT;

    signal    wr_mem : std_logic;
    signal    mem_addr : std_logic_vector(ADDR_WIDTH-1 DOWNTO 0);
    signal    fresh_cmd1 : STD_LOGIC_VECTOR(1 DOWNTO 0);
    signal fresh_k_ti_i : std_logic_vector(1 downto 0);
    signal sel_mux : STD_LOGIC_VECTOR(1 DOWNTO 0);
    signal re_fifo2, wr_fifo2 : std_logic;
    signal mem_data_out : STD_LOGIC_VECTOR(DATA_WIDTH-1 DOWNTO 0);
    signal fifo2_data_out : STD_LOGIC_VECTOR(DATA_WIDTH-1 DOWNTO 0);
    signal fifo2_empty1, fifo2_full1 : std_logic;

begin

    zm_data_out <= tpm_data_in;

    Inst_data_control_top: data_control_top PORT MAP(
        clk => clk ,
        rst => rst ,
        reconfig => reconfig ,
        cpu_cmd => cpu_cmd ,
        tpm_empty => tpm_empty ,
        fifo_empty => fifo_empty ,
        fifo2_empty => fifo2_empty1 ,
        re_tpm => re_tpm ,
        wr_tpm => wr_tpm ,
        re_fifo => re_fifo ,
        wr_fifo => wr_fifo ,
        re_fifo2 => re_fifo2 ,
        wr_fifo2 => wr_fifo2 ,
        wr_mem => wr_mem ,
        mem_addr => mem_addr ,
        fresh_cmd => fresh_cmd1,
        fresh_k_ti_i => fresh_k_ti_i,
        IO_finish => io_finish,
```

```
        ready => ready,
        sel_mux => sel_mux
    );

    Inst_mux3_1by2sel: mux3_1by2sel PORT MAP(
        a => zm_data_in,
        b => fifo2_data_out,
        c => mem_data_out,
        d => GAOZU,
        o => tpm_data_out,
        se => sel_mux
    );

    fifo2_data_control : fifo_2
        port map (
            clk => clk ,
            din => tpm_data_in,
            rd_en => re_fifo2,
            srst => rst,
            wr_en => wr_fifo2,
            dout => fifo2_data_out,
            empty => fifo2_empty1,
            full => fifo2_full1
    );
    Inst_tesla_mem: tesla_mem PORT MAP(
        clk => clk ,
        rst => rst ,
        reconfig => reconfig ,
        wr => wr_mem,
        addr => mem_addr,
        data_in => tpm_data_in,
        data_out => mem_data_out,
        oneway_finish => verify_finish,
        fresh_cmd => fresh_cmd1,
        fresh_k_ti_i => fresh_k_ti_i
    );

end RTL;
```

--- data_control_top.vhd

```
entity data_control_top is
    Port (
            clk : in    STD_LOGIC;
```

```
            rst     : in    STD_LOGIC;
            reconfig : in std_logic;
            --en : in    STD_LOGIC;
        -- 使能信号线
            cpu_cmd : in    STD_LOGIC_VECTOR(CONTROL_CMD_WIDTH-1 downto 0);
                -- 命令内容
            tpm_empty : in    STD_LOGIC;
        -- TPM 空信号
            fifo_empty : in    STD_LOGIC;
        -- FIFO 空信号
            fifo2_empty : in std_logic;                                    --
FIFO2 空信号
            re_tpm : out    STD_LOGIC;
--  从 TPM 读
            wr_tpm : out    STD_LOGIC;
        --  写入 TPM
            re_fifo : out    STD_LOGIC;                                    --
从 fifo 读
            wr_fifo : out    STD_LOGIC;                                    --
写入 fifo
            re_fifo2 : out std_logic;                                      --  主 要
存放解密完珊笮枰  ?
            wr_fifo2 : out std_logic;                                      --  主 要
存放解密完成后需要继续加密的数据
            wr_mem    : out std_logic;                                     --
写入 MEM
            mem_addr : out std_logic_vector(ADDR_WIDTH-1 DOWNTO 0);
        -- MEM ADDRESS
            --fresh_cmd_in : in std_logic_vector(1 downto 0);              --      从
CPU 来的更新命令
            fresh_cmd : out std_logic_vector(1 downto 0);                  --      给
MEM 的更新命令
            fresh_k_ti_i : out std_logic_vector(1 downto 0);               --     更    新
K_Ti_i 的命令 ，给 MEM
            IO_finish : out    STD_LOGIC;
        -- IO 完成
--              --verify_finish : out STD_LOGIC;                           --
单向函数完成(验证 K<Ti-1> 完成)
            ready : out    STD_LOGIC;
        -- 空闲信号
            sel_mux          :      out std_logic_vector( 1 downto 0)      --  数 据
选择信号

        --01：  fifo -> TPM
```

```
	--10:	fifo2 -> TPM

	--11：	MEM -> TPM
			);
end data_control_top;

architecture RTL of data_control_top is

	COMPONENT tpm_fifo_rw
	PORT(
		rst : IN std_logic;
		clk : IN std_logic;
		en : IN std_logic;
		re : IN std_logic;
		tpm_empty : IN std_logic;
		fifo_empty : IN std_logic;
		re_tpm : OUT std_logic;
		wr_tpm : OUT std_logic;
		re_fifo : OUT std_logic;
		wr_fifo : OUT std_logic;
		IO_finish : OUT std_logic;
		ready : OUT std_logic
		);
	END COMPONENT;

	COMPONENT tpm_tesla_init
	PORT(
		clk : IN std_logic;
		rst : IN std_logic;
		en : IN std_logic;
		re : IN std_logic;
		re_tpm : OUT std_logic;
		wr_tpm : OUT std_logic;
		wr_mem : OUT std_logic;
		mem_addr : OUT std_logic_vector(3 downto 0);
		IO_finish : OUT std_logic;
		ready : OUT std_logic
		);
	END COMPONENT;

	COMPONENT tpm_tesla_rw
	PORT(
		clk : IN std_logic;
```

```
        rst : IN std_logic;
        reconfig : IN std_logic;
        en : IN std_logic;
        re : IN std_logic;
        re_tpm : OUT std_logic;
        wr_tpm : OUT std_logic;
        wr_mem : OUT std_logic;
        mem_addr : OUT std_logic_vector(3 downto 0);
        IO_finish : OUT std_logic;
        ready : OUT std_logic;
        fresh_k_ti_i : out std_logic_vector(1 downto 0);
        fresh_cmd : OUT std_logic_vector(1 downto 0)
        );
    END COMPONENT;

    COMPONENT k_ti_oneway_rw
    PORT(
        clk : IN std_logic;
        rst : IN std_logic;
        en : IN std_logic;
        re : IN std_logic;
        re_tpm : OUT std_logic;
        wr_tpm : OUT std_logic;
        wr_mem : OUT std_logic;
        mem_addr : OUT std_logic_vector(3 downto 0);
        IO_finish : OUT std_logic;
        fresh_k_ti_i : out std_logic_vector(1 downto 0);
        ready : OUT std_logic
        );
    END COMPONENT;

    COMPONENT k_ti_fresh
    PORT(
        clk : IN std_logic;
        rst : IN std_logic;
        en : IN std_logic;
        re : IN std_logic;
        re_tpm : OUT std_logic;
        wr_tpm : OUT std_logic;
        wr_mem : OUT std_logic;
        mem_addr : OUT std_logic_vector(3 downto 0);
        IO_finish : OUT std_logic;
        ready : OUT std_logic
        );
```

```
    END COMPONENT;

    COMPONENT tesla_e_to_tpm
    PORT(
        clk : IN std_logic;
        rst : IN std_logic;
        en : IN std_logic;
        re : IN std_logic;
        re_tpm : OUT std_logic;
        wr_tpm : OUT std_logic;
        wr_mem : OUT std_logic;
        mem_addr : OUT std_logic_vector(3 downto 0);
        IO_finish : OUT std_logic;
        ready : OUT std_logic;
        fresh_cmd : OUT std_logic_vector(1 downto 0)
        );
    END COMPONENT;

    COMPONENT read_invalid_data
    PORT(
        rst : IN std_logic;
        clk : IN std_logic;
        en : IN std_logic;
        re : in std_logic;
        tpm_empty : IN std_logic;
        re_tpm : OUT std_logic;
        wr_tpm : OUT std_logic;
        IO_finish : OUT std_logic;
        ready : OUT std_logic
        );
    END COMPONENT;

type state is (wait_state, read_data_state, write_data_state, read_data_to_fifo2_state,
                write_data_from_fifo2_state,read_tesla_init_state,
                write_tesla_init_state,read_tesla_state,write_i_state,
                read_k_t_state,write_k_t_state,write_new_k_t_state,
                write_e_k_i_state,read_invalid_data_state,write_null_data_state);
signal curr_state, next_state : state;
signal en1, en2, en3, en4, en5, en6, en7, en8 : std_logic;
signal re1,re2, re3, re4, re5, re6, re7, re8 : std_logic;
signal re_tpm1, re_tpm2, re_tpm3, re_tpm4, re_tpm5, re_tpm6, re_tpm7, re_tpm8 : std_logic;
signal wr_tpm1, wr_tpm2, wr_tpm3, wr_tpm4, wr_tpm5, wr_tpm6, wr_tpm7, wr_tpm8 :
std_logic;
signal re_fifo1, re_fifo22 : std_logic;
```

```
signal wr_fifo1, wr_fifo22 : std_logic;
signal io_finish1, io_finish2, io_finish3, io_finish4, io_finish5, io_finish6, io_finish7, io_finish8 :
std_logic;
signal ready1, ready2, ready3, ready4, ready5, ready6, ready7, ready8 : std_logic;
signal wr_mem1, wr_mem2, wr_mem3, wr_mem4, wr_mem5 : std_logic;
signal mem_addr1, mem_addr2, mem_addr3, mem_addr4, mem_addr5 :
std_logic_vector(ADDR_WIDTH-1 DOWNTO 0);
signal fresh_cmd1, fresh_cmd2 : std_logic_vector(1 downto 0);
signal fresh_k_ti_i1, fresh_k_ti_i2 : std_logic_vector(1 downto 0);
begin

    Inst_tpm_fifo: tpm_fifo_rw PORT MAP(
        rst => rst ,
        clk => clk ,
        en => en1 ,
        re => re1 ,
        tpm_empty => tpm_empty ,
        fifo_empty => fifo_empty ,
        re_tpm => re_tpm1 ,
        wr_tpm => wr_tpm1 ,
        re_fifo => re_fifo1 ,
        wr_fifo => wr_fifo1 ,
        IO_finish => io_finish1 ,
        ready => ready1
    );

    Inst_tpm_fifo2: tpm_fifo_rw PORT MAP(
        rst => rst ,
        clk => clk ,
        en => en2 ,
        re => re2 ,
        tpm_empty => tpm_empty ,
        fifo_empty => fifo2_empty ,
        re_tpm => re_tpm2 ,
        wr_tpm => wr_tpm2 ,
        re_fifo => re_fifo22 ,
        wr_fifo => wr_fifo22 ,
        IO_finish => io_finish2 ,
        ready => ready2
    );

    Inst_tpm_tesla_init: tpm_tesla_init PORT MAP(
        clk => clk ,
        rst => rst ,
```

```
        en => en3 ,
        re => re3 ,
        re_tpm => re_tpm3 ,
        wr_tpm => wr_tpm3 ,
        wr_mem => wr_mem1 ,
        mem_addr => mem_addr1 ,
        IO_finish => io_finish3 ,
        ready => ready3
);

Inst_tpm_tesla_rw: tpm_tesla_rw PORT MAP(
        clk => clk ,
        rst => rst ,
        reconfig => reconfig ,
        en => en4 ,
        re => re4 ,
        re_tpm => re_tpm4 ,
        wr_tpm => wr_tpm4 ,
        wr_mem => wr_mem2 ,
        mem_addr => mem_addr2 ,
        IO_finish => io_finish4 ,
        ready => ready4 ,
        fresh_k_ti_i => fresh_k_ti_i1,
        fresh_cmd => fresh_cmd1
);

Inst_k_ti_oneway_rw: k_ti_oneway_rw PORT MAP(
        clk => clk ,
        rst => rst ,
        en => en5 ,
        re => re5 ,
        re_tpm => re_tpm5 ,
        wr_tpm => wr_tpm5 ,
        wr_mem => wr_mem3 ,
        mem_addr => mem_addr3 ,
        IO_finish => io_finish5 ,
        fresh_k_ti_i => fresh_k_ti_i2 ,
        ready => ready5
);

Inst_k_ti_fresh: k_ti_fresh PORT MAP(
        clk => clk ,
        rst => rst ,
        en => en6 ,
```

```
        re => re6 ,
        re_tpm => re_tpm6 ,
        wr_tpm => wr_tpm6 ,
        wr_mem => wr_mem4 ,
        mem_addr => mem_addr4 ,
        IO_finish => io_finish6 ,
        ready => ready6
);

Inst_tesla_e_to_tpm: tesla_e_to_tpm PORT MAP(
        clk => clk ,
        rst => rst ,
        en => en7 ,
        re => re7 ,
        re_tpm => re_tpm7 ,
        wr_tpm => wr_tpm7 ,
        wr_mem => wr_mem5 ,
        mem_addr => mem_addr5 ,
        IO_finish => io_finish7 ,
        ready => ready7 ,
        fresh_cmd => fresh_cmd2
);

Inst_read_invalid_data: read_invalid_data PORT MAP(
        rst => rst ,
        clk => clk ,
        en => en8 ,
        re => re8 ,
        tpm_empty => tpm_empty ,
        re_tpm => re_tpm8 ,
        wr_tpm => wr_tpm8 ,
        IO_finish => io_finish8 ,
        ready => ready8
);

process(rst, clk)
begin
        if rst = '1' then
                curr_state <= wait_state;
        elsif clk'event and clk='1' then
                curr_state <= next_state;
        end if;
end process;
```

```
process(curr_state, cpu_cmd,
            en1, en2, en3, en4, en5, en6, en7, en8,
            re1,re2, re3, re4, re5, re6, re7, re8,
            re_tpm1, re_tpm2, re_tpm3, re_tpm4,
            re_tpm5, re_tpm6, re_tpm7, re_tpm8,
            wr_tpm1, wr_tpm2, wr_tpm3, wr_tpm4,
            wr_tpm5, wr_tpm6, wr_tpm7, wr_tpm8,
            re_fifo1, re_fifo22,wr_fifo1, wr_fifo22,
            io_finish1, io_finish2, io_finish3, io_finish4,
            io_finish5, io_finish6, io_finish7, io_finish8,
            ready1, ready2, ready3, ready4,
            ready5, ready6, ready7, ready8,
            wr_mem1, wr_mem2, wr_mem3, wr_mem4, wr_mem5,
            mem_addr1, mem_addr2, mem_addr3, mem_addr4, mem_addr5,
            fresh_cmd1, fresh_cmd2, fresh_k_ti_i1, fresh_k_ti_i2)
begin
case curr_state is
-------------------------------------------------------
-- 等待状态
-------------------------------------------------------
when wait_state =>
      en1 <= '0';
      en2 <= '0';
      en3 <= '0';
      en4 <= '0';
      en5 <= '0';
      en6 <= '0';
      en7 <= '0';
      en8   <= '0';
      re1 <= '0';
      re2 <= '0';
      re3 <= '0';
      re4 <= '0';
      re5 <= '0';
      re6 <= '0';
      re7   <= '0';
      re8 <= '0';
      re_tpm <= '0';
      wr_tpm <= '0';
      re_fifo <= '0';
      wr_fifo <= '0';
      re_fifo2 <= '0';
      wr_fifo2 <= '0';
      wr_mem <= '0';
```

```
        mem_addr <= "0000";
        IO_finish <= '0';
        fresh_k_ti_i <= "00";
        --verify_finish <= '0';
        ready <= '1';
        sel_mux <= "00";
        fresh_cmd <= "00";
        case cpu_cmd is
        when READ_DATA    => next_state <= read_data_state;
        when WRITE_DATA => next_state <= write_data_state;
        when READ_DATA_TO_FIFO2 =>    next_state <= read_data_to_fifo2_state;
        when WRITE_DATA_FROM_FIFO2 =>    next_state <= write_data_from_fifo2_state;
        when READ_TESLA_INIT =>    next_state <= read_tesla_init_state;
        when WRITE_TESLA_INIT =>    next_state <= write_tesla_init_state;
        when READ_TESLA =>    next_state <= read_tesla_state;
        when WRITE_I => next_state <= write_i_state;
        when READ_K_T => next_state <= read_k_t_state;
        when WRITE_K_T =>    next_state <= write_k_t_state;
        when WRTIT_NEW_K_T => next_state <= write_new_k_t_state;
        when WRITE_E_K_I =>    next_state <= write_e_k_i_state;
        when READ_INVALID_DATA_CMD =>    next_state <= read_invalid_data_state;
        when WRITE_NULL_CMD                =>    next_state <= write_null_data_state;
        when others =>    next_state <= wait_state;
        end case;
-------------------------------------------------------
-- 把数据 写入 fifo
-------------------------------------------------------
when read_data_state =>
        en1 <= '1';
        en2 <= '0';
        en3 <= '0';
        en4 <= '0';
        en5 <= '0';
        en6 <= '0';
        en7 <= '0';
        en8 <= '0';
        re1 <= '1';
        re2 <= '0';
        re3 <= '0';
        re4 <= '0';
        re5 <= '0';
        re6 <= '0';
        re7 <= '0';
        re8 <= '0';
```

```
        re_tpm <= re_tpm1;
        wr_tpm <= wr_tpm1;
        re_fifo <= re_fifo1;
        wr_fifo <= wr_fifo1;
        re_fifo2 <= '0';
        wr_fifo2 <= '0';
        wr_mem <= '0';
        mem_addr <= "0000";
        IO_finish <= io_finish1;
        --verify_finish <= '0';
        ready <= ready1;
        sel_mux <= "01";
        fresh_cmd <= "00";
        fresh_k_ti_i <= "00";
        if ( io_finish1 = '1' )then
            next_state <= wait_state;
        else
            next_state <= read_data_state;
        end if;
-------------------------------------------------------
-- 从 FIFO 中写数据入 TPM
-------------------------------------------------------
when write_data_state =>
        en1 <= '1';
        en2 <= '0';
        en3 <= '0';
        en4 <= '0';
        en5 <= '0';
        en6 <= '0';
        en7 <= '0';
        en8 <= '0';
        re1 <= '0';
        re2 <= '0';
        re3 <= '0';
        re4 <= '0';
        re5 <= '0';
        re6 <= '0';
        re7 <= '0';
        re8 <= '0';
        re_tpm <= re_tpm1;
        wr_tpm <= wr_tpm1;
        re_fifo <= re_fifo1;
        wr_fifo <= wr_fifo1;
        re_fifo2 <= '0';
```

```
        wr_fifo2 <= '0';
        wr_mem <= '0';
        mem_addr <= "0000";
        IO_finish <= io_finish1;
        fresh_k_ti_i <= "00";
        --verify_finish <= '0';
        ready <= ready1;
        sel_mux <= "01";
        fresh_cmd <= "00";
        if ( io_finish1 = '1' )then
            next_state <= wait_state;
        else
            next_state <= write_data_state;
        end if;
-------------------------------------------------------
-- 从 TPM 中读取数据进入 fifo2
-------------------------------------------------------
when read_data_to_fifo2_state =>
        en1 <= '0';
        en2 <= '1';
        en3 <= '0';
        en4 <= '0';
        en5 <= '0';
        en6 <= '0';
        en7 <= '0';
        en8 <= '0';
        re1 <= '0';
        re2 <= '1';
        re3 <= '0';
        re4 <= '0';
        re5 <= '0';
        re6 <= '0';
        re7 <= '0';
        re8 <= '0';
        re_tpm <= re_tpm2;
        wr_tpm <= wr_tpm2;
        re_fifo <= '0';
        wr_fifo <= '0';
        re_fifo2 <= re_fifo22;
        wr_fifo2 <= wr_fifo22;
        wr_mem <= '0';
        mem_addr <= "0000";
        IO_finish <= io_finish2;
        --verify_finish <= '0';
```

```
        ready <= ready2;
        sel_mux <= "10";
        fresh_cmd <= "00";
        fresh_k_ti_i <= "00";
        if ( io_finish2 = '1' )then
            next_state <= wait_state;
        else
            next_state <= read_data_to_fifo2_state;
        end if;
-----------------------------------------------------
-- 将 fifo2 中数据 读入 TPM
-----------------------------------------------------
when write_data_from_fifo2_state =>
        en1 <= '0';
        en2 <= '1';
        en3 <= '0';
        en4 <= '0';
        en5 <= '0';
        en6 <= '0';
        en7 <= '0';
        en8 <= '0';
        re1 <= '0';
        re2 <= '0';
        re3 <= '0';
        re4 <= '0';
        re5 <= '0';
        re6 <= '0';
        re7 <= '0';
        re8 <= '0';
        re_tpm <= re_tpm2;
        wr_tpm <= wr_tpm2;
        re_fifo <= '0';
        wr_fifo <= '0';
        re_fifo2 <= re_fifo22;
        wr_fifo2 <= wr_fifo22;
        wr_mem <= '0';
        mem_addr <= "0000";
        IO_finish <= io_finish2;
        fresh_k_ti_i <= "00";
        --verify_finish <= '0';
        ready <= ready2;
        sel_mux <= "10";
        fresh_cmd <= "00";
        if ( io_finish2 = '1' )then
```

```
            next_state <= wait_state;
        else
            next_state <= write_data_from_fifo2_state;
        end if;
-------------------------------------------------------
-- 将 TESLA INIT 的 解密结果读入 MEM
-------------------------------------------------------
when read_tesla_init_state =>
        en1 <= '0';
        en2 <= '0';
        en3 <= '1';
        en4 <= '0';
        en5 <= '0';
        en6 <= '0';
        en7 <= '0';
        en8 <= '0';
        re1 <= '0';
        re2 <= '0';
        re3 <= '1';
        re4 <= '0';
        re5 <= '0';
        re6 <= '0';
        re7 <= '0';
        re8 <= '0';
        re_tpm <= re_tpm3;
        wr_tpm <= wr_tpm3;
        re_fifo <= '0';
        wr_fifo <= '0';
        re_fifo2 <= '0';
        wr_fifo2 <= '0';
        wr_mem <= wr_mem1;
        mem_addr <= mem_addr1;
        IO_finish <= io_finish3;
        fresh_k_ti_i <= "00";
        --verify_finish <= ;
        ready <= ready3;
        sel_mux <= "00";
        fresh_cmd <= "00";
        if ( io_finish3 = '1' )then
            next_state <= wait_state;
        else
            next_state <= read_tesla_init_state;
        end if;
-------------------------------------------------------
```

```
-- 将 拆开的 TESLA INIT 写入 TPM
-----------------------------------------------------
when write_tesla_init_state =>
        en1 <= '0';
        en2 <= '0';
        en3 <= '1';
        en4 <= '0';
        en5 <= '0';
        en6 <= '0';
        en7 <= '0';
        en8 <= '0';
        re1 <= '0';
        re2 <= '0';
        re3 <= '0';
        re4 <= '0';
        re5 <= '0';
        re6 <= '0';
        re7 <= '0';
        re8 <= '0';
        re_tpm <= re_tpm3;
        wr_tpm <= wr_tpm3;
        re_fifo <= '0';
        wr_fifo <= '0';
        re_fifo2 <= '0';
        wr_fifo2 <= '0';
        wr_mem <= wr_mem1;
        mem_addr <= mem_addr1;
        IO_finish <= io_finish3;
        fresh_k_ti_i <= "00";
        --verify_finish <= ;
        ready <= ready3;
        sel_mux <= "11";
        fresh_cmd <= "00";
        if ( io_finish3 = '1' )then
                next_state <= wait_state;
        else
                next_state <= write_tesla_init_state;
        end if;
-----------------------------------------------------
-- 从 TPM 读取 TESLA 到 MEM
-----------------------------------------------------
when read_tesla_state =>
        en1 <= '0';
        en2 <= '0';
```

```
        en3 <= '0';
        en4 <= '1';
        en5 <= '0';
        en6 <= '0';
        en7 <= '0';
        en8 <= '0';
        re1 <= '0';
        re2 <= '0';
        re3 <= '0';
        re4 <= '1';
        re5 <= '0';
        re6 <= '0';
        re7 <= '0';
        re8 <= '0';
        re_tpm <= re_tpm4;
        wr_tpm <= wr_tpm4;
        re_fifo <= '0';
        wr_fifo <= '0';
        re_fifo2 <= '0';
        wr_fifo2 <= '0';
        wr_mem <= wr_mem2;
        mem_addr <= mem_addr2;
        IO_finish <= io_finish4;
        --verify_finish <= ;
        ready <= ready4;
        sel_mux <= "00";
        fresh_cmd <= fresh_cmd1;
        fresh_k_ti_i <= fresh_k_ti_i1;
        if ( io_finish4 = '1' )then
                next_state <= wait_state;
        else
                next_state <= read_tesla_state;
        end if;
-----------------------------------------------------
-- 将 TESLA 需要的 i 送入 TPM
-----------------------------------------------------
when write_i_state =>
        en1 <= '0';
        en2 <= '0';
        en3 <= '0';
        en4 <= '1';
        en5 <= '0';
        en6 <= '0';
        en7 <= '0';
```

```
        en8 <= '0';
        re1 <= '0';
        re2 <= '0';
        re3 <= '0';
        re4 <= '0';
        re5 <= '0';
        re6 <= '0';
        re7 <= '0';
        re8 <= '0';
        re_tpm <= re_tpm4;
        wr_tpm <= wr_tpm4;
        re_fifo <= '0';
        wr_fifo <= '0';
        re_fifo2 <= '0';
        wr_fifo2 <= '0';
        wr_mem <= wr_mem2;
        mem_addr <= mem_addr2;
        IO_finish <= io_finish4;
        fresh_k_ti_i <= fresh_k_ti_i1;
        --verify_finish <= ;
        ready <= ready4;
        sel_mux <= "11";
        fresh_cmd <= fresh_cmd1;
        if ( io_finish4 = '1' )then
            next_state <= wait_state;
        else
            next_state <= write_i_state;
        end if;
-------------------------------------------------------
-- 将 每次 单向 K_T 的结果读入 MEM
-------------------------------------------------------
when read_k_t_state =>
        en1 <= '0';
        en2 <= '0';
        en3 <= '0';
        en4 <= '0';
        en5 <= '1';
        en6 <= '0';
        en7 <= '0';
        en8 <= '0';
        re1 <= '0';
        re2 <= '0';
        re3 <= '0';
        re4 <= '0';
```

```
        re5 <= '1';
        re6 <= '0';
        re7 <= '0';
        re8 <= '0';
        re_tpm <= re_tpm5;
        wr_tpm <= wr_tpm5;
        re_fifo <= '0';
        wr_fifo <= '0';
        re_fifo2 <= '0';
        wr_fifo2 <= '0';
        wr_mem <= wr_mem3;
        mem_addr <= mem_addr3;
        IO_finish <= io_finish5;
        fresh_k_ti_i <= fresh_k_ti_i2;
        ready <= ready5;
        sel_mux <= "00";
        fresh_cmd <= "00";
        if ( io_finish5 = '1' )then
                next_state <= wait_state;
        else
                next_state <= read_k_t_state;
        end if;
-------------------------------------------------------
-- 将 每次单向的 K_T_I 写入 TPM
-------------------------------------------------------
when write_k_t_state =>
        en1 <= '0';
        en2 <= '0';
        en3 <= '0';
        en4 <= '0';
        en5 <= '1';
        en6 <= '0';
        en7 <= '0';
        en8 <= '0';
        re1 <= '0';
        re2 <= '0';
        re3 <= '0';
        re4 <= '0';
        re5 <= '0';
        re6 <= '0';
        re7 <= '0';
        re8 <= '0';
        re_tpm <= re_tpm5;
        wr_tpm <= wr_tpm5;
```

```
        re_fifo <= '0';
        wr_fifo <= '0';
        re_fifo2 <= '0';
        wr_fifo2 <= '0';
        wr_mem <= wr_mem3;
        mem_addr <= mem_addr3;
        IO_finish <= io_finish5;
        fresh_k_ti_i <= fresh_k_ti_i2;
        --verify_finish <= ;
        ready <= ready5;
        sel_mux <= "11";
        fresh_cmd <= "00";
        if ( io_finish5 = '1' )then
            next_state <= wait_state;
        else
            next_state <= write_k_t_state;
        end if;
-----------------------------------------------------
-- 将更新的 K_T_I 写入 TPM
-----------------------------------------------------
when write_new_k_t_state =>
        en1 <= '0';
        en2 <= '0';
        en3 <= '0';
        en4 <= '0';
        en5 <= '0';
        en6 <= '1';
        en7 <= '0';
        en8 <= '0';
        re1 <= '0';
        re2 <= '0';
        re3 <= '0';
        re4 <= '0';
        re5 <= '0';
        re6 <= '0';
        re7 <= '0';
        re8 <= '0';
        re_tpm <= re_tpm6;
        wr_tpm <= wr_tpm6;
        re_fifo <= '0';
        wr_fifo <= '0';
        re_fifo2 <= '0';
        wr_fifo2 <= '0';
        wr_mem <= wr_mem4;
```

```
        mem_addr <= mem_addr4;
        IO_finish <= io_finish6;
        fresh_k_ti_i <= "00";
        --verify_finish <= ;
        ready <= ready6;
        sel_mux <= "11";
        fresh_cmd <= "00";
        if ( io_finish6 = '1' )then
            next_state <= wait_state;
        else
            next_state <= write_new_k_t_state;
        end if;
-------------------------------------------------------
-- 将 E( ( ) ) 慈?TPM
-------------------------------------------------------
when write_e_k_i_state =>
        en1 <= '0';
        en2 <= '0';
        en3 <= '0';
        en4 <= '0';
        en5 <= '0';
        en6 <= '0';
        en7 <= '1';
        en8 <= '0';
        re1 <= '0';
        re2 <= '0';
        re3 <= '0';
        re4 <= '0';
        re5 <= '0';
        re6 <= '0';
        re7    <= '0';
        re8 <= '0';
        re_tpm <= re_tpm7;
        wr_tpm <= wr_tpm7;
        re_fifo <= '0';
        wr_fifo <= '0';
        re_fifo2 <= '0';
        wr_fifo2 <= '0';
        wr_mem <= wr_mem5;
        mem_addr <= mem_addr5;
        IO_finish <= io_finish7;
        fresh_k_ti_i <= "00";
        --verify_finish <= ;
        ready <= ready7;
```

```
        sel_mux <= "11";
        fresh_cmd <= fresh_cmd2;
        if ( io_finish7 = '1' )then
            next_state <= wait_state;
        else
            next_state <= write_e_k_i_state;
        end if;
-----------------------------------------------------
-- 将无用数据读出 TPM
-----------------------------------------------------
when read_invalid_data_state    =>
        en1 <= '0';
        en2 <= '0';
        en3 <= '0';
        en4 <= '0';
        en5 <= '0';
        en6 <= '0';
        en7 <= '0';
        en8    <= '1';
        re1 <= '0';
        re2 <= '0';
        re3 <= '0';
        re4 <= '0';
        re5 <= '0';
        re6 <= '0';
        re7 <= '0';
        re8 <= '1';
        re_tpm <= re_tpm8;
        wr_tpm <= wr_tpm8;
        re_fifo <= '0';
        wr_fifo <= '0';
        re_fifo2 <= '0';
        wr_fifo2 <= '0';
        wr_mem <= '0';
        mem_addr <= "0000";
        IO_finish <= io_finish8;
        fresh_k_ti_i <= "00";
        --verify_finish <= ;
        ready <= ready8;
        sel_mux <= "00";
        fresh_cmd <= "00";
        if ( io_finish8 = '1' )then
            next_state <= wait_state;
        else
```

```
                next_state <= read_invalid_data_state;
            end if;
        when write_null_data_state    =>
            en1 <= '0';
            en2 <= '0';
            en3 <= '0';
            en4 <= '0';
            en5 <= '0';
            en6 <= '0';
            en7 <= '0';
            en8    <= '1';
            re1 <= '0';
            re2 <= '0';
            re3 <= '0';
            re4 <= '0';
            re5 <= '0';
            re6 <= '0';
            re7 <= '0';
            re8 <= '0';
            re_tpm <= re_tpm8;
            wr_tpm <= wr_tpm8;
            re_fifo <= '0';
            wr_fifo <= '0';
            re_fifo2 <= '0';
            wr_fifo2 <= '0';
            wr_mem <= '0';
            mem_addr <= "0000";
            IO_finish <= io_finish8;
            fresh_k_ti_i <= "00";
            --verify_finish <= ;
            ready <= ready8;
            sel_mux <= "00";
            fresh_cmd <= "00";
            if ( io_finish8 = '1' )then
                next_state <= wait_state;
            else
                next_state <= read_invalid_data_state;
            end if;
        end case;
        end process;

end RTL;
```

--- md5.v

```
`timescale 10ns/1ns



module md5(clk,reset,load_i,fin_i,data_i,data_o,out_fin,hash_gen);

input clk;
input reset;
input load_i;
output out_fin;
output hash_gen;
input fin_i;


//Input must be padded and in little endian mode
input [31:0] data_i;
output [31:0] data_o;

reg hash_gen;
reg out_fin, next_out_fin;
reg [31:0] data_o, next_data_o;

reg [5:0] round64, next_round64;
reg [43:0] t;

reg newtext_i, next_newtext_i;
reg [31:0] ar,br,cr,dr,func_out, next_ar,next_br, next_cr, next_dr;
reg [31:0] A,B,C,D,next_A,next_B, next_C, next_D;
reg [511:0] message, next_message;
reg [2:0] round,next_round;
reg next_generate_hash,generate_hash;
reg hash_generated;

reg [4:0] next_getdata_state, getdata_state;
reg [2:0] next_outdata_state, outdata_state;

//rom process
always @(round64)
begin

    case(round64)
            0: t = 44'hD76AA478070;
            1: t = 44'hE8C7B7560C1;
            2: t = 44'h242070DB112;
```

3: t = 44'hC1BDCEEE163;
4: t = 44'hF57C0FAF074;
5: t = 44'h4787C62A0C5;
6: t = 44'hA8304613116;
7: t = 44'hFD469501167;
8: t = 44'h698098D8078;
9: t = 44'h8B44F7AF0C9;
10: t = 44'hFFFF5BB111A;
11: t = 44'h895CD7BE16B;
12: t = 44'h6B90112207C;
13: t = 44'hFD9871930CD;
14: t = 44'hA679438E11E;
15: t = 44'h49B4082116F;

16: t = 44'hf61e2562051;
17: t = 44'hc040b340096;
18: t = 44'h265e5a510EB;
19: t = 44'he9b6c7aa140;
20: t = 44'hd62f105d055;
21: t = 44'h0244145309A;
22: t = 44'hd8a1e6810EF;
23: t = 44'he7d3fbc8144;
24: t = 44'h21e1cde6059;
25: t = 44'hc33707d609E;
26: t = 44'hf4d50d870E3;
27: t = 44'h455a14ed148;
28: t = 44'ha9e3e90505D;
29: t = 44'hfcefa3f8092;
30: t = 44'h676f02d90E7;
31: t = 44'h8d2a4c8a14C;

32: t = 44'hfffa3942045;
33: t = 44'h8771f6810B8;
34: t = 44'h6d9d612210B;
35: t = 44'hfde5380c17E;
36: t = 44'ha4beea44041;
37: t = 44'h4bdecfa90B4;
38: t = 44'hf6bb4b60107;
39: t = 44'hbebfbc7017A;
40: t = 44'h289b7ec604D;
41: t = 44'heaa127fa0B0;
42: t = 44'hd4ef3085103;
43: t = 44'h04881d05176;
44: t = 44'hd9d4d039049;

```
            45: t = 44'he6db99e50BC;
            46: t = 44'h1fa27cf810F;
            47: t = 44'hc4ac5665172;

            48: t = 44'hf4292244060;
            49: t = 44'h432aff970A7;
            50: t = 44'hab9423a70FE;
            51: t = 44'hfc93a039155;
            52: t = 44'h655b59c306C;
            53: t = 44'h8f0ccc920A3;
            54: t = 44'hffeff47d0FA;
            55: t = 44'h85845dd1151;
            56: t = 44'h6fa87e4f068;
            57: t = 44'hfe2ce6e00AF;
            58: t = 44'ha30143140F6;
            59: t = 44'h4e0811a115D;
            60: t = 44'hf7537e82064;
            61: t = 44'hbd3af2350AB;
            62: t = 44'h2ad7d2bb0F2;
            63: t = 44'heb86d391159;
        endcase
end
//end process rom


//funcs process


reg [31:0] aux31,fr_var,tr_var;
//reg [31:0] rotate1,rotate2;
reg [7:0] s_var;
reg [3:0] nblock;
reg [31:0] message_var[15:0];

always @(t or ar or br or cr or dr or round or message or func_out or message_var[0] or message_var[1] or message_var[2] or message_var[3]
            or message_var[4] or message_var[5] or message_var[6] or message_var[7] or message_var[8] or message_var[9] or message_var[10]
            or message_var[11] or message_var[12] or message_var[13] or message_var[14] or message_var[15])
begin

    message_var[0]=message[511:480];
    message_var[1]=message[479:448];
```

```
    message_var[2]=message[447:416];
    message_var[3]=message[415:384];
    message_var[4]=message[383:352];
    message_var[5]=message[351:320];
    message_var[6]=message[319:288];
    message_var[7]=message[287:256];
    message_var[8]=message[255:224];
    message_var[9]=message[223:192];
    message_var[10]=message[191:160];
    message_var[11]=message[159:128];
    message_var[12]=message[127:96];
    message_var[13]=message[95:64];
    message_var[14]=message[63:32];
    message_var[15]=message[31:0];

    fr_var=0;

    case(round)
        0: fr_var=((br&cr)|(~br&dr));
        1: fr_var=((br&dr)|(cr& (~dr)));
        2: fr_var=(br^cr^dr);
        3: fr_var=(cr^(br|~dr));
    endcase


    tr_var=t[43:12];
    s_var=t[11:4];
    nblock=t[3:0];

    aux31=(ar+fr_var+message_var[nblock]+tr_var);

    //rotate1=aux31 << s_var;
    //rotate2=aux31 >> (32-s_var);
    //func_out=br+(rotate1 | rotate2);
      aux31 = aux31 <<< (s_var%32);
    func_out = br+ aux31;
end
//end process funcs


//process round64FSM
always @(newtext_i or round or round64 or ar or br or cr or dr or generate_hash or func_out or
getdata_state or outdata_state or A or B or C or D)
begin
```

```
    next_ar=ar;
    next_br=br;
    next_cr=cr;
    next_dr=dr;
    next_round64=round64;
    next_round=round;
    hash_generated=0;
    hash_gen = 0;
    if(generate_hash!=0)
    begin
        next_ar=dr;
        next_br=func_out;
        next_cr=br;
        next_dr=cr;
    end

    case(round64)
        0:
        begin
            next_round=0;
            if(generate_hash) next_round64=1;
        end

        15,31,47:
        begin
            next_round=round+1;
            next_round64=round64+1;
        end

        63:
        begin
            next_round=0;
            next_round64=0;
            hash_generated=1;
            hash_gen = 1;
        end

        default: next_round64=round64+1;

endcase


if(newtext_i)
```

```
        begin
           next_ar=32'h67452301;
           next_br=32'hEFCDAB89;
           next_cr=32'h98BADCFE;
           next_dr=32'h10325476;
           next_round=0;
           next_round64=0;
        end

        if(!getdata_state)
        begin
           next_ar=A;
           next_br=B;
           next_cr=C;
           next_dr=D;
        end

end
//end round64FSM process

//regsignal process
always @(posedge clk or posedge reset)
begin

     if(reset)
     begin
        out_fin=0;
        data_o=0;
        message=0;

        ar=32'h67452301;
        br=32'hEFCDAB89;
        cr=32'h98BADCFE;
        dr=32'h10325476;

        getdata_state=0;
        generate_hash=0;

         outdata_state=0;

        round=0;
        round64=0;
        A=32'h67452301;
        B=32'hEFCDAB89;
```

```
        C=32'h98BADCFE;
        D=32'h10325476;

    end
    else
    begin
        out_fin = next_out_fin;
        data_o = next_data_o;
        //message = next_message;
         newtext_i = next_newtext_i;
        ar=next_ar;
        br=next_br;
        cr=next_cr;
        dr=next_dr;

        A=next_A;
        B=next_B;
        C=next_C;
        D=next_D;

        generate_hash=next_generate_hash;
        getdata_state=next_getdata_state;
        outdata_state=next_outdata_state;

         if(newtext_i)
         begin
            message = 0;
         end
         else
         begin
            message = next_message;
         end
        round=next_round;
        round64=next_round64;
    end
end
//end regsignals process


//getdata process

reg [31:0] data_o_var;
reg [511:0] aux;
wire [31:0] A_t,B_t,C_t,D_t;
```

```
assign A_t=dr+A;
assign B_t=func_out+B;
assign C_t=br+C;
assign D_t=cr+D;

always @(newtext_i or A_t or B_t or C_t or D_t or data_i or load_i or getdata_state or generate_hash or hash_generated or message or func_out or A or B or C or D or ar or br or cr or dr or fin_i)
begin

    next_A=A;
    next_B=B;
    next_C=C;
    next_D=D;

    next_generate_hash = generate_hash;
    //next_out_fin = 0;
    next_getdata_state = getdata_state;

    aux = message;
    next_message = message;

    if(newtext_i)
    begin
      next_A=32'h67452301;
      next_B=32'hEFCDAB89;
      next_C=32'h98BADCFE;
      next_D=32'h10325476;
      next_getdata_state=0;
    end

    case(getdata_state)

        0 :
        begin
          if(load_i)
          begin
            aux[511:480]=data_i;
            next_message=aux;
            next_getdata_state=1;
//                aux = data_i;
//                next_message[511:480]= aux;
//              next_getdata_state=1;
```

```
          end
        end

        1 :
        begin
         if(load_i)
         begin
           aux[479:448]=data_i;
           next_message=aux;
           next_getdata_state=2;
//                 aux = data_i;
//                 next_message[479:448]= aux;
//               next_getdata_state=2;
         end
           else if(fin_i)
           begin
            next_message[479:0]                                                        =
480'h8000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000020;
            next_getdata_state=16;
          end
        end

        2 :
        begin
         if(load_i)
         begin
           aux[447:416]=data_i;
           next_message=aux;
           next_getdata_state=3;
//                 aux = data_i;
//                 next_message[447:416]= aux;
//               next_getdata_state=3;
         end
           else if(fin_i)
           begin
            next_message[447:0]                                                        =
448'h8000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000040;
            next_getdata_state=16;
         end
        end

        3 :
```

```
            begin
                if(load_i)
                begin
                    aux[415:384]=data_i;
                    next_message=aux;
                    next_getdata_state=4;
//                          aux = data_i;
//                          next_message[415:384]= aux;
//                      next_getdata_state=4;
                end
                  else if(fin_i)
                  begin
                    next_message[415:0]                                                                =
416'h80000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000060;
                    next_getdata_state=16;
              end
            end

            4 :
            begin
              if(load_i)
              begin
                  aux[383:352]=data_i;
                  next_message=aux;
                  next_getdata_state=5;
//                          aux = data_i;
//                          next_message[383:352]= aux;
//                      next_getdata_state=5;
              end
                  else if(fin_i)
                  begin
                    next_message[383:0]                                                                =
384'h80000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000080;
                    next_getdata_state=16;
              end
            end

            5 :
            begin
              if(load_i)
              begin
                  aux[351:320]=data_i;
```

```
        next_message=aux;
        next_getdata_state=6;
//          aux = data_i;
//          next_message[351:320]= aux;
//        next_getdata_state=5;
      end
        else if(fin_i)
        begin
         next_message[351:0]                                                    =
352'h80000000000000000000000000000000000000000000000000000000000000000000000000
000000000000A0;
         next_getdata_state=16;
      end
     end

     6 :
     begin
      if(load_i)
      begin
        aux[319:288]=data_i;
        next_message=aux;
        next_getdata_state=7;
//          aux = data_i;
//          next_message[319:288]= aux;
//        next_getdata_state=7;
      end
        else if(fin_i)
        begin
         next_message[319:0]                                                    =
320'h80000000000000000000000000000000000000000000000000000000000000000000000000
0000C0;
         next_getdata_state=16;
      end
     end

     7 :
     begin
      if(load_i)
      begin
        aux[287:256]=data_i;
        next_message=aux;
        next_getdata_state=8;
//          aux = data_i;
//          next_message[287:256]= aux;
```

```
//                next_getdata_state=8;
          end
             else if(fin_i)
             begin
               next_message[287:0]                                                                                    =
288'h800000000000000000000000000000000000000000000000000000000000000000000000000000E0;
               next_getdata_state=16;
          end
         end

         8 :
         begin
          if(load_i)
          begin
             aux[255:224]=data_i;
             next_message=aux;
             next_getdata_state=9;
//                 aux = data_i;
//                 next_message[255:224]= aux;
//               next_getdata_state=9;
          end
             else if(fin_i)
             begin
               next_message[255:0]                                                                                    =
256'h8000000000000000000000000000000000000000000000000000000000000100;
               next_getdata_state=16;
          end
         end

         9 :
         begin
          if(load_i)
          begin
             aux[223:192]=data_i;
             next_message=aux;
             next_getdata_state=10;
//                 aux = data_i;
//                 next_message[223:192]= aux;
//               next_getdata_state=10;
          end
             else if(fin_i)
             begin
               next_message[223:0]                                                                                    =
224'h80000000000000000000000000000000000000000000000000000120;
```

```
                next_getdata_state=16;
            end
        end

        10 :
        begin
            if(load_i)
            begin
                aux[191:160]=data_i;
                next_message=aux;
                next_getdata_state=11;
//                  aux = data_i;
//                  next_message[191:160]= aux;
//              next_getdata_state=11;
            end
                else if(fin_i)
                begin
                  next_message[191:0]                                                                    =
192'h800000000000000000000000000000000000000000000140;
                  next_getdata_state=16;
            end
        end

        11 :
        begin
            if(load_i)
            begin
                aux[160:128]=data_i;
                next_message=aux;
                next_getdata_state=12;
//                  aux = data_i;
//                  next_message[160:128]= aux;
//              next_getdata_state=12;
            end
                else if(fin_i)
                begin
                  next_message[159:0] = 160'h8000000000000000000000000000000000000160;
                  next_getdata_state=16;
            end
        end

        12 :
        begin
            if(load_i)
```

```
          begin
            aux[127:96]=data_i;
            next_message=aux;
            next_getdata_state=13;
//                  aux = data_i;
//                  next_message[127:96]= aux;
//              next_getdata_state=13;
          end
            else if(fin_i)
            begin
              next_message[127:0] = 128'h80000000000000000000000000000180;
              next_getdata_state=16;
          end
        end

        13 :
        begin
          if(load_i)
          begin
            aux[95:64]=data_i;
            next_message=aux;
            next_getdata_state=14;
//                  aux = data_i;
//                  next_message[95:64]= aux;
//              next_getdata_state=14;
          end
            else if(fin_i)
            begin
              next_message[95:0] = 96'h800000000000000000001A0;
              next_getdata_state=16;
          end
        end

        14 :
        begin
          if(load_i)
          begin
            aux[63:32]=data_i;
            next_message=aux;
            next_getdata_state=15;
//                  aux = data_i;
//                  next_message[63:32]= aux;
//              next_getdata_state=15;
          end
```

```
            else if(fin_i)
            begin
              next_message[63:0] = 64'h80100000000001C0;
              next_getdata_state=16;
          end
        end

        15 :
        begin
          if(load_i)
          begin
            aux[31:0]=data_i;
            next_message=aux;
            next_getdata_state=16;
            //next_generate_hash=1;
//                  aux = data_i;
//                  next_message[31:0]= aux;
//               next_getdata_state=16;
          end
            else if(fin_i)
            begin
              next_message[31:0] = 32'h801001E0;
              next_getdata_state=16;
            next_generate_hash=1;
          end
        end

        16 :
        begin

          next_generate_hash=1;
          if(hash_generated)
          begin
                next_generate_hash = 0;
                next_A = A_t;
                next_B = B_t;
                next_C = C_t;
                next_D = D_t;
                next_getdata_state = 0;
          end
        end
    endcase
end
//end getdata process
```

```
//outdata process
always @(hash_generated or outdata_state or A_t or B_t or C_t or D_t)
begin

    next_outdata_state = outdata_state;
    next_data_o = 0;

    case(outdata_state)
    0:
    begin
          next_out_fin = 0;
          next_newtext_i = 0;
       if(hash_generated)
        begin
         data_o_var = A_t;
         next_data_o = data_o_var;
         next_outdata_state = 1;
        end
        else
        begin
           data_o_var = 32'h10000000;
         next_data_o = data_o_var;
         next_outdata_state = 0;
       end
    end
    1:
    begin
          next_newtext_i = 0;
          next_out_fin = 0;
        data_o_var=B_t;
        next_data_o = data_o_var;
        next_outdata_state = 2;
    end
    2:
    begin
          next_newtext_i = 0;
          next_out_fin = 0;
        data_o_var=C_t;
        next_data_o=data_o_var;
        next_outdata_state = 3;
    end
    3:
    begin
```

```
            next_newtext_i = 0;
            next_out_fin = 1;
          data_o_var=D_t;
          next_data_o= data_o_var;
          next_outdata_state = 4;
    end
    4:
    begin
            next_newtext_i = 1;
        next_out_fin = 0;
          data_o_var= 32'h00000100;
          next_data_o= data_o_var;
          next_outdata_state = 5;
    end
    5:
    begin
            next_newtext_i = 0;
            next_out_fin = 0;
               data_o_var = 32'h00000101;
          next_data_o= data_o_var;
            next_outdata_state = 0;
    end
     default:
     begin
              next_newtext_i = 0;
          next_out_fin = 0;
              data_o_var = 32'h11110000;
          next_data_o= data_o_var;
              next_outdata_state = 0;
     end
     endcase
end

endmodule
```

外围模块顶层 VHDL 代码 top_peripheral.vhd

```
--------------------------------------------------------------------------------
entity top_peripheral is
      PORT(
            --global signals
            clk : IN std_logic;
            rst : IN std_logic;
            --for check
            check_rx_over: out std_logic;
```

```
--
   --LCD_PS2 ext_ports
   rotary_a : IN std_logic;
   rotary_b : IN std_logic;
   clear_display : IN std_logic;

   ps2_clk : INOUT std_logic;
   ps2_data : INOUT std_logic;

   lcd_d : INOUT std_logic_vector(7 downto 4);
   lcd_rs : OUT std_logic;
   lcd_rw : OUT std_logic;
   lcd_e : OUT std_logic;
   strataflash_oe : OUT std_logic;
   strataflash_ce : OUT std_logic;
   strataflash_we : OUT std_logic;
--
  --RF_top    ext_ports
   RS232_rx : IN std_logic;
   RS232_tx : OUT std_logic;

   RF_serial_rx : IN std_logic;
   RF_serial_tx : OUT std_logic;

--Shared ext_ports
   frame_leds : OUT std_logic_vector(7 downto 0);
--
--ports for interconnection with TPM
--for TPM_rx

TPM_ready: in std_logic;            --level sensitive

clear_ack: in std_logic;

   rx_over_mux : OUT std_logic;    --one period pulse
   frame_type_mux : OUT std_logic_vector(7 downto 0);
   read_from_fifo_mux : IN std_logic;
   dout32_present_n_mux : OUT std_logic;
   dout32_mux : OUT std_logic_vector(31 downto 0);

--for TPM_tx
   TX_ready: out std_logic;          --level sensitive
```

```
        ext_OK: IN STD_LOGIC;               --one period pulse
          ext_frame_type: in std_logic_vector(7 downto 0);

          TPM_tx_data: in std_logic_VECTOR(31 downto 0);
        TPM_write_to_fifo: in std_logic;


          --TO TPM for reconfig and send ID signals
        change_happened: out std_logic;     --one period pulse
        RF_mode: out std_logic                  --level sensitive
          );
end top_peripheral;

architecture Behavioral of top_peripheral is


--ps2_lcd module

    COMPONENT top_ps2_lcd
    PORT(
        clk : IN std_logic;
        rst : IN std_logic;
        --
        --PS2 ext_ports
        ps2_clk : INOUT std_logic;
        ps2_data : INOUT std_logic;

        lcd_d : INOUT std_logic_vector(7 downto 4);
        lcd_rs : OUT std_logic;
        lcd_rw : OUT std_logic;
        lcd_e : OUT std_logic;
        strataflash_oe : OUT std_logic;
        strataflash_ce : OUT std_logic;
        strataflash_we : OUT std_logic;

        rotary_a : IN std_logic;
        rotary_b : IN std_logic;
        clear_display : IN std_logic;

        --
        --ports of inter_module interactive

        --from RF_control module for the display changes of RF_modes: BASE or END mode,
```

```
            change_mode : IN std_logic; --one period pulse

          clear_ack: in std_logic;

            --to TPM for encrypted, one of two sources of TPM_rx,need to be mux_ed
          over_info: out std_logic;
            frame_type_ps2: out std_logic_vector(7 downto 0);

            read_from_fifo1vs8 : IN std_logic;
            dout32_present_n : OUT std_logic;
            dout32 : OUT std_logic_vector(31 downto 0)
            );
        END COMPONENT;
--signals for above module

--RF_top module
        COMPONENT RF_top
        PORT(
            clk : IN std_logic;
            rst : IN std_logic;
          --
            --RF_top ext_ports
            RS232_rx : IN std_logic;
            RS232_tx : OUT std_logic;

            RF_serial_rx : IN std_logic;
            RF_serial_tx : OUT std_logic;
            --
            --ports of inter_module interactive
          --ports for TPM_tx
            ext_OK : IN std_logic;
            ext_frame_type : IN std_logic_vector(7 downto 0);

           ext_data: in std_logic_vector(7 downto 0);
           ext_data_present: in std_logic;
           read_from_ext: out std_logic;

          --ports for TPM_rx,which is one of two soures ,need to be muxed to TPM
            payload_over : OUT std_logic;
            frame_type : OUT std_logic_vector(7 downto 0);

            read_from_fifo1vs4_protocol : IN std_logic;
            dout32_present_n_protocl : OUT std_logic;
            dout32_protocol : OUT std_logic_vector(31 downto 0);
```

```
            --
            --ports connect to TPM

            --ports to TPM ,for the situation of this module
            TX_ready: out std_logic;
          --ports TO TPM for reconfig and send ID signals
            IDj_change : OUT std_logic;--these two change_signals need to be muxed out to TPM in top module
            change_mode: out std_logic;--to PS2_LCD for display of RF_module,to mux_rx_to_TPM for mux contents
            RF_mode : OUT std_logic
            );
      END COMPONENT;
-- signals for above module
     signal change_mode: std_logic;
      signal IDj_change :    std_logic;
      signal RF_mode_i : std_logic;


--convert32tx_to_8tx component
   COMPONENT convert_32tx_to_8tx
   port(clk: in std_logic;
          rst: in std_logic;
          --from TPM, TPM write 32 bits to the fifo ,which is readed as 8bits
            TPM_tx_data: in   std_logic_VECTOR(31 downto 0);
            TPM_write_to_fifo: in std_logic;
          --output 8 bits datas to protocol_tx module of RF_top
            ext_data: out std_logic_vector(7 downto 0);
          ext_data_present: out std_logic;
          read_from_ext: in std_logic
         );
      END COMPONENT;
--singals for above module
      signal      ext_data : std_logic_vector(7 downto 0);
      signal      ext_data_present : std_logic;
      signal      read_from_ext :    std_logic;

--MUX_rx_to_TPM module
      COMPONENT MUX_rx_to_TPM
      PORT(
         clk: in std_logic;
         rst: in std_logic;
            --for check
            check_rx_over: out std_logic;
```

```
            --from TPM
            TPM_ready: in std_logic;
          --from RF_top
            change_mode: in std_logic;

          frame_leds: out std_logic_vector(7 downto 0);
            --from PS2
            over_info : IN std_logic;
            frame_type_ps2 : IN std_logic_vector(7 downto 0);
            dout32_present_n : IN std_logic;
            dout32 : IN std_logic_vector(31 downto 0);
            read_from_fifo1vs8 : OUT std_logic;
            --from RF_top
            payload_over : IN std_logic;
            frame_type : IN std_logic_vector(7 downto 0);
            dout32_present_n_protocl : IN std_logic;
            dout32_protocol : IN std_logic_vector(31 downto 0);
            read_from_fifo1vs4_protocol : OUT std_logic;
           --To TPM
            read_from_fifo_mux : IN std_logic;
            dout32_present_n_mux : OUT std_logic;
            dout32_mux : OUT std_logic_vector(31 downto 0);
            rx_over_mux : OUT std_logic;
            frame_type_mux : OUT std_logic_vector(7 downto 0)
            );
      END COMPONENT;
--signals for above module
      signal     over_info :    std_logic;
      signal     frame_type_ps2 :    std_logic_vector(7 downto 0);
      signal     dout32_present_n : std_logic;
      signal     dout32 :    std_logic_vector(31 downto 0);
      signal     read_from_fifo1vs8 : std_logic;

      signal     dout32_present_n_protocl :    std_logic;
      signal     dout32_protocol :    std_logic_vector(31 downto 0);
      signal     payload_over :    std_logic;
      signal     frame_type :    std_logic_vector(7 downto 0);
      signal     read_from_fifo1vs4_protocol :    std_logic;

begin

--Inst ps2_lcd module
      Inst_top_ps2_lcd: top_ps2_lcd PORT MAP(
```

```
            clk => clk,
            rst => rst,
            --
            --ext ports
            ps2_clk => ps2_clk,
            ps2_data => ps2_data,

            lcd_d => lcd_d,
            lcd_rs => lcd_rs,
            lcd_rw => lcd_rw,
            lcd_e => lcd_e,
            strataflash_oe => strataflash_oe,
            strataflash_ce => strataflash_ce,
            strataflash_we => strataflash_we,

            rotary_a => rotary_a,
            rotary_b =>rotary_b ,
            clear_display =>clear_display,

            change_mode => change_mode,

            clear_ack => clear_ack,

            over_info => over_info,
            dout32_present_n => dout32_present_n,
            read_from_fifo1vs8 => read_from_fifo1vs8,
            dout32 => dout32,
            frame_type_ps2 => frame_type_ps2
        );
--
--logic for ps2_lcd module


--Inst RF_top module
        Inst_RF_top: RF_top PORT MAP(
            clk => clk,
            rst => rst,

            RS232_rx => RS232_rx,
            RS232_tx => RS232_tx,

            RF_serial_tx => RF_serial_tx,
            RF_serial_rx => RF_serial_rx,
```

```
            payload_over => payload_over,
            frame_type => frame_type,
        read_from_fifo1vs4_protocol => read_from_fifo1vs4_protocol,
            dout32_present_n_protocl => dout32_present_n_protocl,
            dout32_protocol => dout32_protocol,

            ext_OK => ext_OK,
            ext_frame_type => ext_frame_type,

           ext_data => ext_data,
            ext_data_present => ext_data_present,
            read_from_ext => read_from_ext,

            TX_ready=> TX_ready,

          RF_mode => RF_mode_i,

            IDj_change => IDj_change,
            change_mode => change_mode
        );
--
--logic for RF_top module

RF_mode <= RF_mode_i;

change_happened <= change_mode OR IDj_change;

--Inst convet32_to_8
            Inst_convert_32tx_to_8tx: convert_32tx_to_8tx PORT MAP(
            clk => clk,
            rst => rst,

            ext_data => ext_data,
            ext_data_present => ext_data_present,
            read_from_ext => read_from_ext,

            TPM_tx_data => TPM_tx_data,
          TPM_write_to_fifo => TPM_write_to_fifo
        );
--
--logic for above module


--Inst MUX_rx_to_TPM
```

```
        Inst_MUX_rx_to_TPM: MUX_rx_to_TPM PORT MAP(
            clk => clk,
            rst => rst,

              check_rx_over => check_rx_over,

                TPM_ready => TPM_ready,

                change_mode => RF_mode_i,

                frame_leds => frame_leds,

                over_info => over_info,
                read_from_fifo1vs8 => read_from_fifo1vs8,
                dout32_present_n => dout32_present_n,
                dout32 => dout32,
                frame_type_ps2 => frame_type_ps2,

                read_from_fifo1vs4_protocol => read_from_fifo1vs4_protocol,
                dout32_present_n_protocl => dout32_present_n_protocl,
                dout32_protocol => dout32_protocol,
                payload_over => payload_over,
                frame_type => frame_type,

                read_from_fifo_mux => read_from_fifo_mux,
                dout32_present_n_mux => dout32_present_n_mux,
                dout32_mux => dout32_mux,
                rx_over_mux => rx_over_mux,
                frame_type_mux => frame_type_mux
        );
--
--logic for above module
end Behavioral;
```

RF 模块顶层文件 RF_top.vhd-

```
----------------------------------------------------------------------------------
entity RF_top is
    port( clk: in std_logic;
                rst: in std_logic;
                    --
                    --ext ports
                    RS232_rx: in std_logic;
                    RS232_tx: out std_logic;
```

```
                RF_serial_tx: out std_logic;
                RF_serial_rx: in    std_logic;

--              leds: out std_logic_vector(5 downto 0);

                --from    protocol_rx to TPM,need to be muxed
               payload_over : OUT std_logic;
               frame_type : OUT std_logic_vector(7 downto 0);
               read_from_fifo1vs4_protocol : IN std_logic;
               dout32_present_n_protocl : OUT std_logic;
               dout32_protocol : OUT std_logic_vector(31 downto 0);

                --for TPM to protocol_tx
                TX_ready: out std_logic;

                ext_OK: in std_logic;
               ext_frame_type: in std_logic_vector(7 downto 0);
                ext_data: in std_logic_vector(7 downto 0);
               ext_data_present: in std_logic;
               read_from_ext: out std_logic;

                --TO TPM for reconfig and send ID signals
               RF_mode: out std_logic;

                IDj_change : out std_logic;
                change_mode: out std_logic
            );
end RF_top;

architecture Behavioral of RF_top is

--RF_mode_control component
    COMPONENT RF_mode_control
    PORT(
        clk : IN std_logic;
        rst : IN std_logic;
      --ext_ports
--      leds : OUT std_logic_vector(5 downto 0);

        RS232_rx : IN std_logic;
        RS232_tx : OUT std_logic;

      --in ports to set RF_mode ,from protocol_tx,reconfig_counter,to set RF_mode
        change_to_base : IN std_logic;
```

```
            unicast_to_base : IN std_logic;
            broad_to_end : IN std_logic;
            broad_to_base : IN std_logic;
            --in singals to stimulate sending the HI<IDIDIDID> packet
            send_HI_ID: IN std_logic;
            --in ports from RF_rx to send datas to COM, may need to be changed for debug
            RF_rx_fifo_in_com : IN std_logic_vector(7 downto 0);
            write_to_RF_rx_fifo_com : IN std_logic;
            --out port for RF_tx to transmit datas
            write_to_RF: out std_logic;
           RF_tx_data : out std_logic_vector(7 downto 0);
            RF_tx_full : in std_logic;       --for tx checking
            -- tx control
            RF_tx_allowed_n: in std_logic;--for tx checking
            RF_tx_state: in std_logic;       --connect to data_present port of RF_tx fifo,for checking
tx state
         RF_tx_over: out std_logic         --outport for tx controlling
            );
      END COMPONENT;
--signals for RF_mode_control

            signal write_to_RF_com: std_logic :='0';
            signal RF_tx_data_com: std_logic_vector(7 downto 0);
          signal RF_tx_over: std_logic :='0';



            signal RF_rx_fifo_in_com: std_logic_vector(7 downto 0);
            signal write_to_RF_rx_fifo_com: std_logic;

            signal change_to_base: std_logic :='0';
            signal unicast_to_base: std_logic :='0';
            signal broad_to_end: std_logic :='0';
            signal broad_to_base: std_logic :='0';
           signal change_to_base_event: std_logic :='0';
            signal BASE_start: std_logic:='0';
--
--RF_interface component
      COMPONENT RF_interface
      PORT(
            clk : IN std_logic;
            rst : IN std_logic;
            --ext_ports
            RF_serial_rx : IN std_logic;
            RF_serial_tx : OUT std_logic;
```

```
        --
        --ports for inner_module interactive
           --ports for RF_tx
           RF_tx_data : IN std_logic_vector(7 downto 0);
           write_to_RF : IN std_logic;
           RF_tx_full : OUT std_logic;
           RF_tx_fifo_data_present: out std_logic;
           read : OUT std_logic;
        --ports for RF_rx
           read_from_RF : IN std_logic;
           RF_rx_data : OUT std_logic_vector(7 downto 0);
           RF_data_present : OUT std_logic;
           RF_rx_full : OUT std_logic
           );
     END COMPONENT;
---- signal for RF_interface
           signal RF_tx_data: std_logic_vector(7 downto 0);
           signal RF_tx_full: std_logic :='0';
           signal write_to_RF: std_logic :='0';
           signal TPM_read: std_logic :='0';
          signal RF_tx_state: std_logic := '0';

           signal RF_rx_data: std_logic_vector(7 downto 0);
           signal RF_data_present: std_logic :='0';
           signal read_from_RF: std_logic :='0';
--
--
--protocol_rx component
     COMPONENT protocol_rx
     PORT(
          clk : IN std_logic;
          rst : IN std_logic;

        -- from RF_rx_fifo
          rx_data_present : IN std_logic;
          receive_data : IN std_logic_vector(7 downto 0);
          read_from_rx : OUT std_logic;

        change_to_base: in std_logic;

         IDj_change         : out std_logic;

          command_ack : OUT std_logic;
          reconfig_counter_rst : OUT std_logic;
```

```
            write_rx_to_ex : OUT std_logic;
            payload : OUT std_logic_vector(7 downto 0);
            payload_over : OUT std_logic;

            frame_type : OUT std_logic_vector(7 downto 0);

            data_out_to_fifo : OUT std_logic_vector(7 downto 0);
            write_to_fifo : OUT std_logic
            );
        END COMPONENT;
--signals for protocol_rx
            signal reconfig_counter_rst: std_logic:='0';

            signal write_rx_to_ex: std_logic :='0';
            signal payload: std_logic_vector(7 downto 0);


        COMPONENT make_rx_4bytes
        PORT(
            clk : IN std_logic;
            rst : IN std_logic;
            write_rx_to_ex : IN std_logic;
            payload : IN std_logic_vector(7 downto 0);

            read_from_fifo1vs4_protocol : IN std_logic;
            dout32_present_n_protocl : OUT std_logic;
            dout32_protocol : OUT std_logic_vector(31 downto 0)
            );
        END COMPONENT;

----signals of TPM need to be implemented here
--
--protocol_tx component
        COMPONENT protocol_tx
        PORT(
            clk : IN std_logic;
            rst : IN std_logic;

            ext_data : IN std_logic_vector(7 downto 0);
            ext_data_present : IN std_logic;
           read_from_ext : OUT std_logic;

            --ports for tx_control
```

```
            ext_allowed : IN std_logic;
            ext_frame_type : IN std_logic_vector(7 downto 0);
            protocol_tx_over: out std_logic;

            --ports to RF_tx_fifo
            data_in : OUT std_logic_vector(7 downto 0);
            write_tx_fifo : OUT std_logic       ;
            tx_fifo_full : IN std_logic;
            read : IN std_logic


            );
      END COMPONENT;
--signals for protocol_tx
            signal TPM_data_in: std_logic_vector(7 downto 0);
            signal TPM_write_tx_fifo: std_logic:='0';
           signal protocol_tx_over: std_logic;
--signals of TPM need need to be implemented here

--tx_control component
      COMPONENT tx_control
      PORT(
            clk : IN std_logic;
            rst : IN std_logic;
           --ports for MUX of two sources of RF_tx_data
            TPM_data_in: in std_logic_vector(7 downto 0);
           TPM_write_tx_fifo: in std_logic;

           RF_tx_data_com: in std_logic_vector(7 downto 0);
           write_to_RF_com: in std_logic;

           RF_tx_data: out std_logic_vector(7 downto 0);
           write_to_RF: out std_logic;

            --from TPM
            ext_OK : IN std_logic;
            ext_frame_type : IN std_logic_vector(7 downto 0);

          TX_ready: out std_logic;

            --from RF_mode_control
            RF_control_tx_over : IN std_logic;
            --for RF_mode_control
            unicast_to_base : OUT std_logic;
```

```
            broad_to_end : OUT std_logic;
            broad_to_base : OUT std_logic;
            RF_tx_allowed_n: out std_logic;

            --from protocol_tx
            protocol_tx_over: in std_logic;
            --for protocol_tx
            frame_type: out std_logic_vector(7 downto 0);
            ext_allowed : OUT std_logic
            );
        END COMPONENT;
--signals of tx_control
          signal ext_allowed: std_logic:='0';
          signal ext_frame_type_latched:    std_logic_vector(7 downto 0);
          signal RF_tx_allowed_n:    std_logic :='0';

--counter component
        COMPONENT counter
        PORT(
            clk : IN std_logic;
            rst : IN std_logic;
            BASE : IN std_logic;
            CE : IN std_logic;
            reconfig_counter_full : OUT std_logic;
            send_HI_ID : OUT std_logic
            );
        END COMPONENT;
--signals of counter
          signal counter_rst: std_logic:='0';
            signal CE: std_logic:='1';
            signal BASE: std_logic :='0';
            signal reconfig_counter_full: std_logic:='0';
            signal send_HI_ID: std_logic:='0';

begin

--inst_RF_mode_control
        Inst_RF_mode_control: RF_mode_control PORT MAP(
            clk => clk,
            rst => rst,

--          leds => leds,
            RS232_rx => RS232_rx,
            RS232_tx => RS232_tx,
```

```
        write_to_RF => write_to_RF_com,
         RF_tx_data    => RF_tx_data_com,
          RF_tx_full    => RF_tx_full,
          -- RF 发送 MUX 判断信?
          RF_tx_allowed_n    => RF_tx_allowed_n,
          RF_tx_state => RF_tx_state,
        RF_tx_over => RF_tx_over,


          RF_rx_fifo_in_com => RF_rx_fifo_in_com,
          write_to_RF_rx_fifo_com => write_to_RF_rx_fifo_com,

          change_to_base => change_to_base,
          unicast_to_base => unicast_to_base,
          broad_to_end => broad_to_end,
          broad_to_base => broad_to_base,
          send_HI_ID => send_HI_ID
      );
--
--logic for RF_mode_control

change_to_base <= reconfig_counter_full;

RF_mode <= BASE;

check_reconfig_counter:process(clk)
begin
   if rising_edge(clk) then
      if rst='1' then
        BASE <='0';
      else
          BASE_start <='0';
        if reconfig_counter_full='1' then
          change_to_base_event <='1';
            CE<='0';
        end if;
        if change_to_base_event='1' and RF_tx_over='1' then --
          BASE <='1';
            CE <='1';
            BASE_start<='1';
            change_to_base_event<='0';
        end if;
       end if;
```

```
  end if;
end process check_reconfig_counter;

   change_mode<= BASE_start;
--
--Inst RF_interface
      Inst_RF_interface: RF_interface PORT MAP(
            clk => clk,
            rst => rst,

            RF_serial_tx => RF_serial_tx,
            RF_serial_rx => RF_serial_rx,

            RF_tx_full => RF_tx_full,
            RF_tx_data => RF_tx_data,
            write_to_RF => write_to_RF,
            read => TPM_read,
            RF_tx_fifo_data_present => RF_tx_state,


            RF_rx_data => RF_rx_data,
            RF_data_present => RF_data_present,
            read_from_RF => read_from_RF,
            RF_rx_full => open
      );
--
----logic for RF_interface
--
--
--Inst protocol_rx
      Inst_protocol_rx: protocol_rx PORT MAP(
            clk => clk,
            rst => rst,

            --connect protocol_rx directly to RF_rx
            rx_data_present => RF_data_present,
            receive_data => RF_rx_data,
            read_from_rx => read_from_RF,

            command_ack => open,
            reconfig_counter_rst => reconfig_counter_rst,

            --ports for TPM
            write_rx_to_ex => write_rx_to_ex,
```

```
            payload => payload,
            payload_over => payload_over,

            frame_type => frame_type,

            data_out_to_fifo => RF_rx_fifo_in_com,
            write_to_fifo => write_to_RF_rx_fifo_com,

          change_to_base => change_to_base,

           IDj_change         =>IDj_change
        );
--
--logic for protocol_rx


        Inst_make_rx_4bytes: make_rx_4bytes PORT MAP(
            clk => clk,
            rst => rst,

            write_rx_to_ex => write_rx_to_ex,
            payload => payload,

            read_from_fifo1vs4_protocol => read_from_fifo1vs4_protocol,
            dout32_present_n_protocl => dout32_present_n_protocl,
            dout32_protocol => dout32_protocol
        );
--
--Inst protocol_tx
        Inst_protocol_tx: protocol_tx PORT MAP(
            clk => clk,
            rst => rst,

            ext_data => ext_data,
            ext_data_present => ext_data_present,
            read_from_ext => read_from_ext,

            ext_allowed => ext_allowed,
            ext_frame_type => ext_frame_type_latched,

            protocol_tx_over => protocol_tx_over,
            --
            data_in => TPM_data_in,
            write_tx_fifo => TPM_write_tx_fifo,
```

```
            tx_fifo_full => RF_tx_full,
            read => TPM_read
        );
--
--logic for protocol_tx


--Inst tx_control
        Inst_tx_control: tx_control PORT MAP(
            clk => clk,
            rst => rst,
          --from protocol_tx,need to be muxed
            TPM_data_in => TPM_data_in,
            TPM_write_tx_fifo => TPM_write_tx_fifo,
            --from com_tx, need to be muxed
            RF_tx_data_com => RF_tx_data_com,
            write_to_RF_com => write_to_RF_com,
            --the muxed signals to RF_tx
            RF_tx_data => RF_tx_data,
            write_to_RF =>write_to_RF,
          --signals for controlling of tx
            RF_control_tx_over => RF_tx_over,
            protocol_tx_over => protocol_tx_over,
            --from TPM ,which need to be latched
            ext_OK => ext_OK,
            ext_frame_type => ext_frame_type,
          --the latched signals to TPM
            ext_allowed => ext_allowed,
            frame_type => ext_frame_type_latched,
          --to RF_mode_control ,to set RF_mode
            RF_tx_allowed_n => RF_tx_allowed_n,

            unicast_to_base => unicast_to_base,
            broad_to_end => broad_to_end,
            broad_to_base => broad_to_base,
          --out to TPM
            TX_ready=>TX_ready
        );
--
--logic for tx_control

--Inst write_RF——control_rx_fifo
--      Inst_write_RF_control_rx_fifo: write_RF_control_rx_fifo PORT MAP(
--            clk => clk,
```

```
--            rst => rst,
--
--            --signals from protocol_rx
--            command_ack => command_ack,
--            --data write to RF_mode_control_RF_rx fifo
--            data_out_to_fifo => RF_rx_fifo_in_com,
--            write_to_fifo => write_to_RF_rx_fifo_com
--     );
--
--logic for above module

--Inst counter
       Inst_counter: counter PORT MAP(
              clk => clk,
              rst => counter_rst,
              reconfig_counter_full => reconfig_counter_full,
              send_HI_ID => send_HI_ID,
              BASE => BASE,
              CE => CE
       );
--
--     logic for counter
counter_rst <= rst OR reconfig_counter_rst OR BASE_start;

end Behavioral;
```

RF 模式控制模块 RF_mode_control.vhd

```
----------------------------------------------------------------------------------
entity RF_mode_control is
     port( clk : in std_logic;
                rst : in std_logic;

                   write_to_RF: out std_logic;
                   RF_tx_data : out std_logic_vector(7 downto 0);
                   RF_tx_full : in std_logic;

                   RF_tx_allowed_n: in std_logic; --高电平禁?
                   RF_tx_state: in std_logic;          --用于检测 RFFIFO 是否为空
               RF_tx_over: out std_logic;          --输出 1 周期高电平

                   RF_rx_fifo_in_com: in    std_logic_vector(7 downto 0);
                   write_to_RF_rx_fifo_com: in std_logic;

                  change_to_base    : in std_logic ;
```

```
                unicast_to_base : in std_logic ;
                broad_to_end       : in std_logic ;
                broad_to_base      : in std_logic ;
                 send_HI_ID          : in std_logic ;

--                leds : out std_logic_vector(5 downto 0);

                 RS232_rx: in std_logic;                                    --from and to computer
                 RS232_tx: out std_logic
              );
end RF_mode_control;

architecture Behavioral of RF_mode_control is


    --PROGRAM rom
    COMPONENT net_comm
        PORT
            (
                address         : IN std_logic_vector(9 downto 0);
              clk               : IN std_logic;
              instruction : OUT std_logic_vector(17 downto 0);
              proc_reset    : OUT std_logic
             );
        END COMPONENT;

        -- ps2_Program memory signals
        signal address          : std_logic_vector (9 downto 0);
        signal PRO_RST          : std_logic ;

    COMPONENT kcpsm3
        PORT(
              instruction     : IN std_logic_vector(17 downto 0);
              in_port           : IN std_logic_vector(7 downto 0);
             interrupt        : IN std_logic;
             reset              : IN std_logic;
             clk                 : IN std_logic;
              address          : OUT std_logic_vector(9 downto 0);
              port_id           : OUT std_logic_vector(7 downto 0);
              write_strobe   : OUT std_logic;
              out_port         : OUT std_logic_vector(7 downto 0);
             read_strobe     : OUT std_logic;
             interrupt_ack : OUT std_logic
             );
```

```
    END COMPONENT;

    -- PicoBlaze Signals
    signal instruction    : std_logic_vector (17 downto 0);
    signal port_id        : std_logic_vector (7 downto 0);
    signal write_strobe : std_logic;
    signal in_port        : std_logic_vector (7 downto 0);
    signal out_port       : std_logic_vector (7 downto 0);
    signal read_strobe    : std_logic;
  signal KCP_RST          : std_logic;
    signal interrupt      : std_logic :='0';
    signal interrupt_event: std_logic :='0';
    signal interrupt_ack: std_logic ;
    signal interrupt_kind: std_logic_vector(7 downto 0) :=(others =>'0');

--UART_rx MODULE
    COMPONENT uart_rx
        PORT
        (
          serial_in                : IN std_logic;
          read_buffer              : IN std_logic;
          reset_buffer             : IN std_logic;
           en_16_x_baud            : IN std_logic;
          clk                      : IN std_logic;
          data_out                 : OUT std_logic_vector(7 downto 0);
          buffer_data_present : OUT std_logic;
           buffer_full             : OUT std_logic;
          buffer_half_full      : OUT std_logic
        );
    END COMPONENT;

    COMPONENT uart_tx
    PORT
    (
        data_in               : IN std_logic_vector(7 downto 0);
        write_buffer        : IN std_logic;
        reset_buffer        : IN std_logic;
         en_16_x_baud       : IN std_logic;
        clk                   : IN std_logic;
        serial_out          : OUT std_logic;
         buffer_full        : OUT std_logic;
        buffer_half_full : OUT std_logic
    );
    END COMPONENT;
```

```
	--UART_rx signals
		signal baud_count		: std_logic_vector (8 downto 0);
		signal en_16_x_baud		: std_logic;
		signal read_from_uart	: std_logic;
		signal rx_data			: std_logic_vector(7 downto 0);
		signal uart_data_present : std_logic;
		signal uart_rx_full		: std_logic;
		signal write_to_uart		: std_logic;
		signal uart_tx_full		: std_logic;
	--LED signals
--		signal write_to_leds		: std_logic;

	COMPONENT RF_rx_fifo
	PORT(
		data_in : IN std_logic_vector(7 downto 0);
		reset : IN std_logic;
		write : IN std_logic;
		read : IN std_logic;
		clk : IN std_logic;
		data_out : OUT std_logic_vector(7 downto 0);
		full : OUT std_logic;
		half_full : OUT std_logic;
		data_present : OUT std_logic
		);
	END COMPONENT;

--
--	--RF signals
		signal read_from_RF		: std_logic;
		signal RF_rx_data		: std_logic_vector(7 downto 0);
		signal RF_data_present : std_logic;
		signal RF_rx_full		: std_logic;

		signal sel			: std_logic_vector(4 downto 0) :="00000";



signal status_port: std_logic_vector(7 downto 0):=(others=>'0');
signal RF_tx_full_new: std_logic :='0';
signal write_to_TX_over: std_logic :='0';


begin
```

```
-- Instantiate PicoBlaze and the instruction ROM.    This is simply
-- cut and paste from the example designs that come with PicoBlaze.
-- Interrupts are not used for this design.


      my_kcpsm3: kcpsm3                        --no interrupt?
          PORT MAP
           (
                address             => address,
                instruction      => instruction,
                port_id             => port_id,
                write_strobe    => write_strobe,
                out_port           => out_port,
                read_strobe      => read_strobe,
                in_port             => in_port,
                interrupt         => interrupt,
                interrupt_ack => interrupt_ack,
                reset                => KCP_RST,
                clk                   => clk
           );

        KCP_RST <= rst OR PRO_RST;
-- without JTAG
      my_program: net_comm
          PORT MAP
           (
               address         => address,
               instruction => instruction,
               proc_reset    => PRO_RST,
                clk                => clk
           );
  -- Interrupt
------------------------------------------------------------------------------------------------------------
  interrupt_control: process(clk)
  begin
    if clk'event and clk='1' then

      -- processor interrupt waits for an acknowledgement
      if interrupt_ack='1' then
          interrupt <= '0';
        elsif interrupt_event='1' then
          interrupt <= '1';
        else
```

```
            interrupt <= interrupt;
        end if;

    end if;
  end process interrupt_control;

 check_interrupt: process(clk)
 begin
   if rising_edge(clk) then
      sel<= send_HI_ID & change_to_base & unicast_to_base & broad_to_end &
broad_to_base;
      interrupt_event <= send_HI_ID OR change_to_base OR unicast_to_base OR
broad_to_end OR broad_to_base;
      case sel is
        when "10000"=> interrupt_kind<="00010010";

          when "01000"=> interrupt_kind<="00001000";
        when "00100"=> interrupt_kind<="00000100";
        when "00010"=> interrupt_kind<="00000010";
          when "00001"=> interrupt_kind<="00000001";
        when others=> interrupt_kind<=interrupt_kind;
      end case;
    end if;

 end process check_interrupt;



  --
  -- Set baud rate to 9600 for the UART communications
  -- Requires en_16_x_baud to be 153600Hz which is a single cycle pulse every 326 cycles at
50MHz
  --
    baudgen: process (clk,rst)
    begin
    if rst = '1' then
        baud_count <= "000000000";
            en_16_x_baud <= '0';
    elsif (clk'event and clk = '1') then
                if (baud_count = 65)then --10Mhz
                  baud_count <= "000000000";
                  en_16_x_baud <= '1';
        else
                  baud_count <= baud_count + 1;
                  en_16_x_baud <= '0';
```

```
            end if;
        end if;
        end process;


      -- Implement the output port logic:
           --      leds_out, port 01
           --      RF_tx, port 02
              --      TX_over,port 10
              --      UART_tx,port 20

        write_to_RF      <= write_strobe and port_id(1);
--      write_to_leds <= write_strobe and port_id(0);
        write_to_uart <= write_strobe and port_id(5);
        write_to_TX_over <= write_strobe and port_id(4);

        RF_tx_data <= out_port;

write_tx_over_status:process(clk)
        begin
         if rising_edge(clk) then
          if(rst='1')then
             RF_tx_over <= '0';
          elsif(write_to_TX_over='1') then
               RF_tx_over <= out_port(0);
                else
                RF_tx_over <='0';
          end if;
         end if;
end process write_tx_over_status;



--write_led: process (clk,rst)
--      begin
--         if (clk'event and clk='1') then
--              if rst = '1'       then
--                leds <= "000000";
--            elsif(write_to_leds = '1') then
--                    leds <= out_port(5 downto 0);
--             end if;
--          end if;
--      end process write_led;
```

```
transmit: uart_tx
 PORT MAP
 (
        data_in                 => out_port,
        write_buffer          => write_to_uart,
        reset_buffer          => rst,
        en_16_x_baud          => en_16_x_baud,
        serial_out             => rs232_tx,
        buffer_full            => uart_tx_full,
        buffer_half_full => open,
        clk                        => clk
 );



 --    Implement the input port logic:
       --    status_port:    port 00
          --uart_tx_full
             --RF_tx_full
              --uart_data_present,
            --RF_data_present,
              --rx_buffer_full,
              --RF_buffer_full,
     --    RF_rx ,        port 04
        --    uart_data_rx, port 08
        -- interrupt_kind, port 40

 process (clk,rst)
 begin
 if (clk'event and clk = '1') then
              if rst = '1' then
              in_port <= "00000000";
              read_from_uart <= '0';
                    read_from_RF      <= '0';
              else
              case (port_id) is
                          when X"00" =>
                                in_port <= status_port;
                          when X"04" =>
                                in_port <= RF_rx_data;
                          when X"08" =>
                                in_port <= rx_data;
                    when X"40" =>
                         in_port <= interrupt_kind;
                          --place other in_ports here
```

```
                        when others =>
                            in_port <= "00000000";
                end case;
            end if;
        -- Form read strobe for receiver FIFO buffer .
        -- The fact that the read strobe will occur after the actual data is read by
        -- the KCPSM3 is acceptable because it is really means 'I have read you'!
                read_from_uart <= read_strobe and port_id(3);
                read_from_RF    <= read_strobe and port_id(2);
               --place other read_port signal here

            end if;
    end process;

    RF_tx_full_new <= RF_tx_full OR RF_tx_allowed_n;

    status_port <= "0"& RF_tx_state & uart_tx_full& RF_tx_full_new & uart_data_present &
RF_data_present
                                & uart_rx_full          & RF_rx_full;

    receive: uart_rx
    PORT MAP
    (
        serial_in               => rs232_rx,
        data_out                  => rx_data,
        read_buffer              => read_from_uart,
        reset_buffer            => rst,
        en_16_x_baud             => en_16_x_baud,
        buffer_data_present => uart_data_present,
        buffer_full              => uart_rx_full,
        buffer_half_full       => open,
        clk                         => clk
    );

    Inst_RF_rx_fifo: RF_rx_fifo PORT MAP(
        data_in => RF_rx_fifo_in_com,
        data_out => RF_rx_data,
        reset => rst,
        write => write_to_RF_rx_fifo_com,
        read => read_from_RF,
        full => RF_rx_full,
        half_full => open,
        data_present => RF_data_present,
```

```
            clk => clk
        );

end Behavioral;
```

发送协议代码 protocol_tx.vhd

```
entity protocol_tx is
    port(      clk: in std_logic;
                rst: in std_logic;
----signals for simulation
--              data: in std_logic_vector(7 downto 0);
--              write_to: in std_logic;
----simulation above

                ext_data: in std_logic_vector(7 downto 0);
                ext_data_present: in std_logic;
                read_from_ext: out std_logic;

                ext_allowed: in std_logic;
                ext_frame_type: in std_logic_vector(7 downto 0);
                protocol_tx_over: out std_logic;

                data_in :out std_logic_vector(7 downto 0);
                write_tx_fifo: out std_logic;
                tx_fifo_full: in std_logic;

                read: in std_logic
          );
end protocol_tx;

architecture Behavioral of protocol_tx is

--constants and signals for frame header
constant start_byte: std_logic_vector(7 downto 0) :=X"7E";
constant address_byte: std_logic_vector(7 downto 0):=X"01";
signal frame_type: std_logic_vector(7 downto 0);
signal length_byte: std_logic_vector(7 downto 0);

--tx_fifo_inner
component tx_bbfifo_inner
        Port (          data_in : in std_logic_vector(7 downto 0);
                        data_out : out std_logic_vector(7 downto 0);
                            reset : in std_logic;
                            write : in std_logic;
```

```
                                read : in std_logic;
                                full : out std_logic;
                           half_full : out std_logic;
                        data_present : out std_logic;
                                 clk : in std_logic);
    end component;

--signals for tx_fifo_inner
signal data_in_inner: std_logic_vector(7 downto 0);
signal data_out_inner: std_logic_vector(7 downto 0);
signal write_to_fifo_inner: std_logic :='0';
signal read_from_fifo_inner: std_logic :='0';
signal fifo_full_inner: std_logic;
signal fifo_data_present_inner: std_logic;

--signal data_reg: std_logic_vector(7 downto 0);
--FSM state definition
type fsm_state is
( idle , ready_to_tx,write_to_fifo,write_over,tx_start,tx_address,tx_type,tx_length,tx_fifo,tx_over
);

signal state: fsm_state := idle;
signal tx_count: std_logic_vector(7 downto 0) :=X"00";
signal state_interrupt: std_logic;
signal data_reg: std_logic_vector(7 downto 0) :=X"00";
signal state_reg_flag: std_logic:='0';
signal data_reg_flag: std_logic:='0';
signal fake_full: std_logic:='0';

begin

manage_FSM: process(clk)
begin
   if(rising_edge(clk)) then
     if rst='1' then
        state <= idle;
     elsif tx_fifo_full = '0' then

            write_to_fifo_inner <='0';
            read_from_ext <='0';
            read_from_fifo_inner <='0';
            write_tx_fifo <='0';
            state_interrupt<='0';
            protocol_tx_over <= '0';
```

```
if state_interrupt='1' and state /= idle then

   if fake_full ='1' then
      fake_full <='0';
   else
     write_tx_fifo <='1';
   end if;

      state <= state;
      state_reg_flag <='0';
      data_reg_flag <='0';
   if data_reg_flag='1' then
     state_reg_flag <='1';
   end if;

else
   if state_reg_flag ='1' then
      state_reg_flag <='0';
      state<= state;

      data_in <=data_reg;
      write_tx_fifo <='1';
      tx_count <= tx_count - 1;
      read_from_fifo_inner <='1';

   else

         case state is

         when idle =>
               if ext_allowed ='1' and ext_data_present ='1' then
                  state <= ready_to_tx;
                  frame_type <= ext_frame_type;
                else
                  state <= idle;
                end if;

         when    ready_to_tx =>
               read_from_ext <='1';
               state <= write_to_fifo;

         when write_to_fifo =>
               if ext_data_present = '1' then
```

```
            data_in_inner <= ext_data;
            write_to_fifo_inner <='1';
            tx_count <= tx_count + 1;
          state <= write_to_fifo;
          read_from_ext <='1';
       else
              state <= write_over;
              read_from_ext <='0';
         end if;

when write_over =>
          length_byte <= tx_count;
          state <= tx_start;

when tx_start =>
          state <= tx_address;
          data_in <= start_byte;
          write_tx_fifo <='1';

when    tx_address =>
          data_in <=address_byte;
          write_tx_fifo <='1';
          state <= tx_type;

when tx_type =>
        data_in <= frame_type;
        write_tx_fifo <='1';
        state <= tx_length;

when tx_length =>
   data_in <= length_byte;
        write_tx_fifo <= '1';
        state <= tx_fifo;
read_from_fifo_inner <='1';
when tx_fifo =>
          data_in <= data_out_inner;
          write_tx_fifo <= '1';
          tx_count <= tx_count - 1;
          read_from_fifo_inner <='1';
          if (tx_count = X"01" ) then
              read_from_fifo_inner <='0';
              tx_count <=X"00";
              state <= tx_over;
          end if;
```

```
                    when tx_over =>
                      state <= idle;
                            protocol_tx_over <= '1';

                    when others =>
                       state <= idle;

                       end case;
               end if;
               end if;
            else
               state_interrupt <= '1';
               write_to_fifo_inner <='0';
               read_from_ext <='0';
               read_from_fifo_inner <='0';
               write_tx_fifo <='0';
               protocol_tx_over <= '0';

               if state_interrupt <='0' and tx_count > X"01" then
                  data_reg <= data_out_inner;
                  data_reg_flag <= '1';
               end if;

            if read='1'and state_interrupt='0' then
                fake_full <='1';
            end if;
        end if;
      end if;
end process manage_FSM;

tx_fifo_inst: tx_bbfifo_inner
   port map (              data_in => data_in_inner,
                            data_out => data_out_inner,
                                reset => rst,
                                write => write_to_fifo_inner,
                                  read => read_from_fifo_inner,
                                  full => fifo_full_inner,
                           half_full => open,
                      data_present => fifo_data_present_inner,
                                    clk => clk);
end Behavioral;
```

接收协议源代码 protocol_rx.vhd

```
----------------------------------------------------------------------------------

entity protocol_rx is
    port( clk: in std_logic;
          rst: in std_logic;

          rx_data_present: in std_logic;
          receive_data     : in std_logic_vector(7 downto 0);
          read_from_rx     : out std_logic;

          command_ack      : out std_logic;
          reconfig_counter_rst: out std_logic;

          change_to_base : in std_logic;

          IDj_change       : out std_logic;

          payload          : out std_logic_vector(7 downto 0);
--        payload_present: out std_logic;
          payload_over     : out std_logic;
          write_rx_to_ex : out std_logic;
--
--        address_byte     : out std_logic_vector(7 downto 0);
--        length_byte      : out std_logic_vector(7 downto 0);
          frame_type       : out std_logic_vector(7 downto 0);

          data_out_to_fifo : OUT std_logic_vector(7 downto 0);
          write_to_fifo : OUT std_logic
        );
end protocol_rx;

architecture Behavioral of protocol_rx is

--PAN_ID,different from each PAN
constant ID: std_logic_vector(7 downto 0):=X"31";

--cosntants and signals used for check the header of one received frame
constant start_byte : std_logic_vector(7 downto 0) := X"7E";
signal rx_counter     : std_logic_vector(7 downto 0):=(others=>'0');
signal IDj: std_logic_vector(7 downto 0) :=X"00";

signal payload_1: std_logic_vector(7 downto 0);
signal write_to_fifo_1:std_logic:='0';
```

```
signal payload_2: std_logic_vector(7 downto 0);
signal write_to_fifo_2:std_logic:='0';

--FSM    states for control of the protocol
type fsm_state is
(           idle,read_first,receive_start,receive_address,receive_type,receive_length,receive_payload,
read_IDj_3,
     receive_O,                                  receive_OK,                                  receive_H,
receive_I,write_IDj_1,write_IDj_2,write_IDj_3,read_IDj_1,read_IDj_2
);
signal state: fsm_state := idle;
signal state_interrupt: std_logic :='0';

type fsm2_state is
(idle2 ,write_ID1, write_ID2, write_ID3,write_ID4
);
signal state2: fsm2_state :=idle2;

signal tx_id: std_logic:='0';
begin

manage_FSM: process(clk,rx_data_present)--,rst,state,receive_data,rx_data_present)
  begin
    if    rising_edge(clk) then
          if rst='1' then
            state <= idle;
            IDj <=X"00";
            elsif rx_data_present = '1' then
          read_from_rx <= '0';
            command_ack    <='0';
            payload_over <='0';
--          payload_present <='0';
          state_interrupt <='0';
            write_to_fifo_1 <='0';
          IDj_change <='0';
          reconfig_counter_rst <='0';
          write_to_fifo <='0';          --signal for RF_mode_control_RF_rx_fifo
            if state_interrupt ='1' and state /= idle then
                    state <= state;
                    read_from_rx <='1';
          else

                case state is
```

```
                when idle =>
                  read_from_rx <='1';
                  state <= read_first;

                when read_first =>
                  if(receive_data=start_byte) then
                      state <= receive_start;
                      read_from_rx <='1';
                   elsif(receive_data=X"4F") then
                      state <= receive_O;
                      read_from_rx <='1';
                      elsif(receive_data=X"48") then
                         state <= receive_H;
                          read_from_rx <='1';
                        else
                          state <= idle;
                          read_from_rx <='0';
                  end if;

                      data_out_to_fifo<=receive_data;
                      write_to_fifo<='1';


                when receive_start =>
                     state <= receive_address;
--                      address_byte <= receive_data;
                      read_from_rx <='1';

                      data_out_to_fifo<=receive_data;
                      write_to_fifo<='1';

                  when receive_address =>
                          state <= receive_type;
                          frame_type <= receive_data;
                      read_from_rx <='1';

                      data_out_to_fifo<=receive_data;
                      write_to_fifo<='1';

                  when receive_type =>
                      state <= receive_length;
--                      length_byte <= receive_data;
                      rx_counter <=  receive_data;
                      read_from_rx <= '1';
```

```
                    data_out_to_fifo<=receive_data;
                    write_to_fifo<='1';

                when receive_length =>
                        state <= receive_payload;
                        payload_1 <=receive_data;
--                      payload_present <='1';
                      write_to_fifo_1 <='1';
                        rx_counter <= rx_counter-1;
                        read_from_rx <='1';
                      data_out_to_fifo<=receive_data;
                      write_to_fifo<='1';

                        if(rx_counter = X"01") then
                        payload_over <='1';
                        state <= idle;
                        read_from_rx <='0';
                        end if;

            when receive_payload =>
                        if(rx_counter = X"01") then
                        payload_1 <=receive_data;
--                      payload_present <= '1';
                        write_to_fifo_1 <='1';
                        payload_over <= '1';
                        rx_counter <= rx_counter -1;
                        state <= idle;
                        read_from_rx <= '0';
                        else
                        payload_1 <= receive_data;
                        write_to_fifo_1 <='1';
--                      payload_present <='1';
                        state <=receive_payload;
                        read_from_rx <= '1';
                        rx_counter <= rx_counter - 1;
                      end if;

                    data_out_to_fifo<=receive_data;
                    write_to_fifo<='1';

            when receive_O =>
                        if(receive_data =X"4B") then
                        state <= receive_OK;
```

```
            read_from_rx <='1';
          else
            state <= idle;
            read_from_rx <='0';
          end if;

          data_out_to_fifo<=receive_data;
          write_to_fifo<='1';



    when receive_OK =>
          if(receive_data=X"0D") then
          command_ack <= '1';
          end if;
          state <= idle;
          read_from_rx <= '0';


          data_out_to_fifo<=receive_data;
          write_to_fifo<='1';

    when receive_H =>
      if(receive_data=X"49") then
         state <= receive_I;
         read_from_rx <='1';
      else
         state <= idle;
         read_from_rx <='0';
      end if;

          data_out_to_fifo<=receive_data;
          write_to_fifo<='1';

when receive_I =>
          reconfig_counter_rst <='1';
            if(IDj/=receive_data) then
              IDj<=receive_data;
              payload_1<= receive_data;
              write_to_fifo_1<='1';
              state <= write_IDj_1;
            else
              state <= read_IDj_1;
            end if;
```

```
        read_from_rx <='1';

            data_out_to_fifo<=receive_data;
            write_to_fifo<='1';

    when write_IDj_1 =>
            payload_1<=receive_data;
            write_to_fifo_1<='1';
            state<=write_IDj_2;
            read_from_rx<='1';

            data_out_to_fifo<=receive_data;
            write_to_fifo<='1';


    when write_IDj_2 =>
       payload_1<=receive_data;
       write_to_fifo_1<='1';
            state<=write_IDj_3;
            read_from_rx<='1';

            data_out_to_fifo<=receive_data;
            write_to_fifo<='1';

    when write_IDj_3 =>
            payload_1<=receive_data;
            write_to_fifo_1<='1';
            state<=idle;
            read_from_rx<='0';
            IDj_change <='1';

            data_out_to_fifo<=receive_data;
            write_to_fifo<='1';

when read_IDj_1 =>
        read_from_rx<='1';
            state<=read_IDj_2;

            data_out_to_fifo<=receive_data;
            write_to_fifo<='1';
when read_IDj_2 =>
        read_from_rx<='1';
            state<=read_IDj_3;
```

```
                        data_out_to_fifo<=receive_data;
                        write_to_fifo<='1';
                when read_IDj_3 =>
                        read_from_rx<='0';
                        state<=idle;

                        data_out_to_fifo<=receive_data;
                        write_to_fifo<='1';


                        when others =>
                            state <=idle;
                            read_from_rx <='0';

                    end case;

                end if;
            else
              state_interrupt <='1';
            read_from_rx <= '0';
              command_ack    <='0';
              payload_over <='0';
--            payload_present <='0';
              write_to_fifo_1 <='0';
            IDj_change <='0';
              reconfig_counter_rst <='0';
              write_to_fifo<='0';
            end if;
         end if;
end process manage_FSM;

manage_FSM2: process(clk)
begin
   if rising_edge(clk) then
        if rst='1' then
           state2<=idle2;
             tx_id <='0';
          else
             tx_id<='0';
              case state2 is

              when idle2 =>
                 if change_to_base='1'       then
                   state2<=write_ID1;
```

```
                tx_id<='1';
                  else
                state2<=idle2;
               end if;

        when write_ID1 =>
                    tx_id<='1';
                       write_to_fifo_2<='1';
                       payload_2<=ID;
                       state2<=write_ID2;

        when write_ID2 =>
                    tx_id<='1';
                       write_to_fifo_2<='1';
                       payload_2<=ID;
                       state2<=write_ID3;

        when write_ID3 =>
                    tx_id<='1';
                       write_to_fifo_2<='1';
                       payload_2<=ID;
                       state2<=write_ID4;

        when write_ID4 =>
                    tx_id<='1';
                       write_to_fifo_2<='1';
                       payload_2<=ID;
                       state2<=idle2;

        when others =>
                     state2<=idle2;


            end case;
        end if;
    end if;
end process manage_FSM2;

MUX:process(tx_id,payload_1,payload_2,write_to_fifo_1,write_to_fifo_2)
begin
        if tx_id='1' then
        payload<=payload_2;
          write_rx_to_ex<=write_to_fifo_2;
        else
```

```
    payload<=payload_1;
     write_rx_to_ex<=write_to_fifo_1;
     end if;
end process MUX;
end Behavioral;
```

2. 汇编代码

网络通信控制代码 net_comm.psm

```
=================================================================
                    Description
=================================================================
                    ;this code is intened for net_comm program
=================================================================
                    ;                   Port address definitions...
=================================================================
                        CONSTANT status, 00                 ; status read port
                    CONSTANT RF_tx_state ,40            ; RF_tx_State      -- bit 6
                    CONSTANT uart_tx_full,20            ;uart tx buffer full -- bit5
                    CONSTANT RF_tx_full ,10             ;RF tx buffer full    -- bit 4
                        CONSTANT uart_data_present, 08    ; uart rx data present --bit 3
                        CONSTANT RF_data_present, 04      ;
                        CONSTANT rx_buffer_full, 02       ; uart rx buffer full    --bit 1
                        CONSTANT RF_buffer_full, 01       ; RF receive buffer full --bit 0
                    CONSTANT RF_data_tx, 02             ; UART transmit write port
                        CONSTANT RF_data_rx, 04             ; RF receive read port
                        CONSTANT TX_over_port,10
                        CONSTANT uart_data_rx, 08           ; UART receive read port
                        CONSTANT uart_data_tx, 20           ; UAT transmit write port

                    CONSTANT interrupt_kind, 40         ; kind of interupt read port
                        CONSTANT change_to_base, 08         ; change to base
                        CONSTANT broad_to_end, 02           ;
                    CONSTANT send_HI_ID_flag,10         ;
                        ;
                                    Special Register usage
*********************************************************************************
                        NAMEREG sF, RF_data
                        NAMEREG sE, NET_para
                    NAMEREG sB, int_kind                ;used to remember the interrupt kind
                    NAMEREG sA, UART_data               ; remember the UART data
*********************************************************************************
                        Scratch Pad Memory Locations
*********************************************************************************
```

```
                CONSTANT RF_mode, 00
                CONSTANT ISR_preserve_s0, 01
================================================================
;                       Useful constant declarations...
================================================================
                CONSTANT all_clear, 00          ; define zero
                ;ASCII table
                CONSTANT ascii_0, 30            ; ascii code
                CONSTANT ascii_1, 31            ; ascii code
                CONSTANT ascii_2, 32            ; ascii code
                CONSTANT ascii_3, 33            ; ascii code
                CONSTANT ascii_4, 34            ; ascii code
                CONSTANT ascii_5, 35            ; ascii code
                CONSTANT ascii_6, 36            ; ascii code
                CONSTANT ascii_7, 37            ; ascii code
                CONSTANT ascii_8, 38            ; ascii code
                CONSTANT ascii_9, 39            ; ascii code
                CONSTANT ascii_A, 41            ; ascii code
                CONSTANT ascii_B, 42            ; ascii code
                CONSTANT ascii_C, 43            ; ascii code
                CONSTANT ascii_D, 44            ; ascii code
                CONSTANT ascii_E, 45            ; ascii code
                CONSTANT ascii_F, 46            ; ascii code
                CONSTANT ascii_G, 47            ; ascii code
                CONSTANT ascii_H, 48            ; ascii code
                CONSTANT ascii_I, 49            ; ascii code
                CONSTANT ascii_J, 4A            ; ascii code
                CONSTANT ascii_K, 4B            ; ascii code
                CONSTANT ascii_L, 4C            ; ascii code
                CONSTANT ascii_M, 4D            ; ascii code
                CONSTANT ascii_N, 4E            ; ascii code
                CONSTANT ascii_O, 4F            ; ascii code
                CONSTANT ascii_P, 50            ; ascii code
                CONSTANT ascii_Q, 51            ; ascii code
                CONSTANT ascii_R, 52            ; ascii code
                CONSTANT ascii_S, 53            ; ascii code
                CONSTANT ascii_T, 54            ; ascii code
                CONSTANT ascii_U, 55            ; ascii code
                CONSTANT ascii_V, 56            ; ascii code
                CONSTANT ascii_W, 57            ; ascii code
                CONSTANT ascii_X, 58            ; ascii code
                CONSTANT ascii_Y, 59            ; ascii code
                CONSTANT ascii_Z, 5A            ; ascii code
                CONSTANT ascii_CARET, 5E        ; ascii code
```

```
                CONSTANT ascii_UNDER, 5F            ; ascii code
                CONSTANT ascii_TICK, 60             ; ascii code
                CONSTANT ascii_a, 61                ; ascii code
                CONSTANT ascii_b, 62                ; ascii code
                CONSTANT ascii_c, 63                ; ascii code
                CONSTANT ascii_d, 64                ; ascii code
                CONSTANT ascii_e, 65                ; ascii code
                CONSTANT ascii_f, 66                ; ascii code
                CONSTANT ascii_g, 67                ; ascii code
                CONSTANT ascii_h, 68                ; ascii code
                CONSTANT ascii_i, 69                ; ascii code
                CONSTANT ascii_j, 6A                ; ascii code
                CONSTANT ascii_k, 6B                ; ascii code
                CONSTANT ascii_l, 6C                ; ascii code
                CONSTANT ascii_m, 6D                ; ascii code
                CONSTANT ascii_n, 6E                ; ascii code
                CONSTANT ascii_o, 6F                ; ascii code
                CONSTANT ascii_p, 70                ; ascii code
                CONSTANT ascii_q, 71                ; ascii code
                CONSTANT ascii_r, 72                ; ascii code
                CONSTANT ascii_s, 73                ; ascii code
                CONSTANT ascii_t, 74                ; ascii code
                CONSTANT ascii_u, 75                ; ascii code
                CONSTANT ascii_v, 76                ; ascii code
                CONSTANT ascii_w, 77                ; ascii code
                CONSTANT ascii_x, 78                ; ascii code
                CONSTANT ascii_y, 79                ; ascii code
                CONSTANT ascii_z, 7A                ; ascii code
                ;Useful constants for network operation
                CONSTANT ID, 31
                CONSTANT MY, 31
                CONSTANT DL, 30
                CONSTANT BASE, 30
                CONSTANT BROADCAST, 46              ;
            ;For 34 MHz clock
            CONSTANT delay_1us_constant,07
 ================================================================
                ; Actual assembly program goes here...
================================================================
    cold_start: LOAD s0, all_clear              ;zero out reg s0
                STORE s0, RF_mode               ;
                CALL RF_reset                   ;initialise the module
                ENABLE INTERRUPT                ;
****************************************************************
```

```
                    ;Main program
*********************************************************************
                    ;Description: control of the network
                    ;
    Wait_for_rx: INPUT sD, status                   ;
                    TEST sD, uart_data_present      ;
                    CALL NZ, read_from_uart         ;
                    TEST sD, RF_data_present        ;
                    CALL NZ, read_from_RF           ;
                    JUMP Wait_for_rx                ;
                    ;
                    ;
  read_from_RF: INPUT RF_data, RF_data_rx           ;
                    LOAD UART_data, RF_data         ;
                CALL out_to_uart                    ;
                RETURN                              ;
                ;
read_from_uart: INPUT UART_data, uart_data_rx       ;
                    LOAD RF_data, UART_data         ;
                    CALL out_to_RF                  ;
                RETURN                              ;
                ;
    out_to_uart: INPUT sD, status                   ;
                TEST sD, uart_tx_full               ;
                JUMP Z, UART_write                  ;
                JUMP out_to_uart                    ;
     UART_write: OUTPUT UART_data, uart_data_tx     ;
                    RETURN                          ;
                   ;
      out_to_RF: INPUT sD, status                   ;
                TEST sD, RF_tx_full                 ;
                JUMP Z, RF_write                    ;
                JUMP out_to_RF                      ;
       RF_write: OUTPUT RF_data, RF_data_tx         ;
                    RETURN                          ;
                    ;
*********************************************************************
                    ;RF Module Routines
*********************************************************************
                    ;Reset the module to its ID, MY, DL
       RF_reset: CALL AT_enter                      ;enter the AT_command mode
                    LOAD    RF_data,    ascii_A                                 ;Send:
ATID<ID>,MY<MY>,DL<DL>,WR,CN<CR>
                    CALL    out_to_RF       ;
```

```
                LOAD RF_data, ascii_T           ;
                CALL    out_to_RF       ;
                LOAD RF_data, ascii_I           ;
                CALL    out_to_RF       ;
                LOAD RF_data, ascii_D           ;
                CALL    out_to_RF       ;
                LOAD RF_data, ID                ;
                CALL    out_to_RF       ;
                LOAD RF_data, ascii_COMMA       ;
                CALL    out_to_RF       ;
                CALL delay_1ms                  ;
                ;
                LOAD RF_data, ascii_M           ;
                CALL    out_to_RF       ;
                LOAD RF_data, ascii_Y           ;
                CALL    out_to_RF       ;
                LOAD RF_data, MY                ;
                CALL    out_to_RF       ;
                LOAD RF_data, ascii_COMMA       ;
                CALL    out_to_RF       ;
                CALL delay_1ms                  ;
                ;
                LOAD RF_data, ascii_D           ;
                CALL    out_to_RF       ;
                LOAD RF_data, ascii_L           ;
                CALL    out_to_RF       ;
                LOAD RF_data, DL                ;
                CALL    out_to_RF       ;
                LOAD RF_data, ascii_COMMA       ;
                CALL    out_to_RF       ;
                CALL delay_1ms                  ;
                ;
Write_quit_AT: LOAD RF_data, ascii_W            ;
                CALL    out_to_RF       ;
                LOAD RF_data, ascii_R           ;
                CALL    out_to_RF       ;
                LOAD RF_data, ascii_COMMA       ;
                CALL    out_to_RF       ;
                CALL delay_60ms                 ;
                ;
                LOAD RF_data, ascii_C           ;
                CALL    out_to_RF       ;
                LOAD RF_data, ascii_N           ;
                CALL    out_to_RF       ;
```

```
                LOAD RF_data, ascii_CR          ;
                CALL    out_to_RF       ;
                RETURN                          ;
                ;
                ;enter the AT_command mode GT + CC + GT
                ;register used s0,s1,s2, RF_data
                ;
        AT_enter: CALL delay_4ms                ;GT > 2ms
                LOAD RF_data, ascii_PLUS        ;transmit "+++"
                CALL    out_to_RF       ;
                CALL    out_to_RF       ;
                CALL    out_to_RF       ;
                CALL delay_4ms                  ;GT > 2ms
                RETURN                          ;
                ;
*************************************************************************
                ;BASE operation routines
*************************************************************************
    Broad_to_end: CALL AT_enter                 ;enter AT_command
                LOAD RF_data, ascii_A
                CALL    out_to_RF       ;
                LOAD RF_data, ascii_T           ;
                CALL    out_to_RF       ;
                LOAD RF_data, ascii_I           ;
                CALL    out_to_RF       ;
                LOAD RF_data, ascii_D           ;
                CALL    out_to_RF       ;
                LOAD RF_data, ID                ;
                CALL    out_to_RF       ;
                LOAD RF_data, ascii_COMMA       ;
                CALL    out_to_RF       ;
                LOAD RF_data, ascii_D           ;
                CALL    out_to_RF       ;
                LOAD RF_data, ascii_L           ;
                CALL    out_to_RF       ;
                LOAD RF_data, ascii_F           ;
                CALL    out_to_RF       ;
                LOAD RF_data, ascii_F           ;
                CALL    out_to_RF       ;
                LOAD RF_data, ascii_F           ;
                CALL    out_to_RF       ;
                LOAD RF_data, ascii_F           ;
                CALL    out_to_RF       ;
                LOAD RF_data, ascii_COMMA       ;
```

```
                CALL    out_to_RF          ;
                CALL Write_quit_AT                  ;
              TEST int_kind, send_HI_ID_flag    ;
              JUMP Z, quit                      ;
                                                ;
send_HI_ID: LOAD RF_data, ascii_H               ;
              CALL    out_to_RF                 ;
              LOAD RF_data, ascii_I             ;
              CALL    out_to_RF                 ;
              LOAD RF_data, ID                  ;
              CALL    out_to_RF                 ;
              LOAD RF_data, ID                  ;
              CALL    out_to_RF                 ;
              LOAD RF_data, ID                  ;
              CALL    out_to_RF                 ;
              LOAD RF_data, ID                  ;
              CALL    out_to_RF                 ;
      quit:     RETURN                          ;
                ;Broadcast to other Base globally
                ;
Broad_to_base: CALL AT_enter                    ;enter AT_command
                LOAD    RF_data,    ascii_A                         ;send
ATDL<BASE>,ID<FFFF>,WR,CN<CR>
                CALL    out_to_RF         ;
                LOAD RF_data, ascii_T             ;
                CALL    out_to_RF         ;
                LOAD RF_data, ascii_D             ;
                CALL    out_to_RF         ;
                LOAD RF_data, ascii_L             ;
                CALL    out_to_RF         ;
                LOAD RF_data, BASE                ;
                CALL    out_to_RF         ;
                LOAD RF_data, ascii_COMMA         ;
                CALL    out_to_RF         ;
                LOAD RF_data, ascii_I             ;
                CALL    out_to_RF         ;
                LOAD RF_data, ascii_D             ;
                CALL    out_to_RF         ;
                LOAD RF_data, ascii_F             ;
                CALL    out_to_RF         ;
                LOAD RF_data, ascii_F             ;
                CALL    out_to_RF         ;
                LOAD RF_data, ascii_F             ;
                CALL    out_to_RF         ;
```

```
                LOAD RF_data, ascii_F           ;
                CALL    out_to_RF       ;
                LOAD RF_data, ascii_COMMA       ;
                CALL    out_to_RF       ;
                CALL Write_quit_AT      ;
                JUMP end_ISR            ;
                ;
*************************************************************************
                ;End device openration routines
*************************************************************************
Unicast_to_base: CALL AT_enter          ;enter AT_command
                LOAD    RF_data,    ascii_A                     ;send
ATDL<BASE>,WR,CN<CR>
                CALL    out_to_RF       ;
                LOAD RF_data, ascii_T           ;
                CALL    out_to_RF       ;
                LOAD RF_data, ascii_D           ;
                CALL    out_to_RF       ;
                LOAD RF_data, ascii_L           ;
                CALL    out_to_RF       ;
                LOAD RF_data, BASE              ;
                CALL    out_to_RF       ;
                LOAD RF_data, ascii_COMMA       ;
                CALL    out_to_RF       ;
                CALL Write_quit_AT      ;
                RETURN                  ;
                ;
                ;
                ;Change_to_base
                ;register used NET_para(sE),RF_data
                ;
 Change_to_base: LOAD NET_para, 01          ;set RF_mode=01
                OUTPUT NET_para, RF_data_tx     ;
                STORE NET_para, RF_mode         ;leds(2 downto 0)='1';
                CALL AT_enter                   ;enter AT_command
                LOAD    RF_data,    ascii_A                     ;send
ATMY<BASE>,DL<FFFF>,WR,CN<CR>
                CALL    out_to_RF       ;
                LOAD RF_data, ascii_T           ;
                CALL    out_to_RF       ;
                LOAD RF_data, ascii_M           ;
                CALL    out_to_RF       ;
                LOAD RF_data, ascii_Y           ;
                CALL    out_to_RF       ;
```

```
                LOAD RF_data, BASE                  ;
                CALL    out_to_RF       ;
                LOAD RF_data, ascii_COMMA       ;
                CALL    out_to_RF       ;
            LOAD RF_data, ascii_D           ;
                CALL    out_to_RF       ;
                LOAD RF_data, ascii_L           ;
                CALL    out_to_RF       ;
                LOAD RF_data, ascii_F           ;
                CALL    out_to_RF           ;
                CALL    out_to_RF           ;
                CALL    out_to_RF           ;
                CALL    out_to_RF           ;
                LOAD RF_data, ascii_COMMA       ;
            CALL out_to_RF                  ;
                CALL Write_quit_AT          ;
                CALL send_HI_ID             ;
            JUMP end_ISR                    ;
*****************************************************************************
                ;Software delay routines
*****************************************************************************
                ;
                ;Delay of 1us.
     delay_1us: LOAD s0, delay_1us_constant
      wait_1us: SUB s0, 01
                JUMP NZ, wait_1us
                RETURN
                ;
                ;Delay of 40us.
                ;
                ;Registers used s0, s1
                ;
    delay_40us: LOAD s1, 28                     ;40 x 1us = 40us
     wait_40us: CALL delay_1us
                SUB s1, 01
                JUMP NZ, wait_40us
                RETURN
                ;
                ;
                ;Delay of 1ms.
                ;
                ;Registers used s0, s1, s2
                ;
     delay_1ms: LOAD s2, 19                     ;25 x 40us = 1ms
```

```
  wait_1ms: CALL delay_40us
            SUB s2, 01
            JUMP NZ, wait_1ms
            RETURN
            ;
            ;
            ;Delay of 2ms
            ;
            ;Registers used s0,s1,s2
 delay_4ms: LOAD s2, 64                    ;100 x 40us = 2ms
  wait_4ms: CALL delay_40us
            SUB s2, 01
            JUMP NZ, wait_1ms
            RETURN
            ;
            ;
            ;Delay of 20ms.
            ;
            ;Delay of 20ms used during initialisation.
            ;
            ;Registers used s0, s1, s2, s3
            ;
delay_20ms: LOAD s3, 14                    ;20 x 1ms = 20ms
 wait_20ms: CALL delay_1ms
            SUB s3, 01
            JUMP NZ, wait_20ms
            RETURN
            ;
            ;
            ;Delay of 60ms
            ;registers used s0,s1,s2,s3
            ;
delay_60ms: LOAD s3, 3C                    ;60 x 1ms = 60ms
 wait_60ms: CALL delay_1ms
            SUB s3, 01
            JUMP NZ, wait_60ms
            RETURN
            ;
            ;
            ;
            ;Delay of approximately 1 second.
            ;
            ;Registers used s0, s1, s2, s3, s4
            ;
```

```
        delay_1s: LOAD s4, 32                       ;50 x 20ms = 1000ms
         wait_1s: CALL delay_20ms
                  SUB s4, 01
                  JUMP NZ, wait_1s
                  RETURN
                  ;
*****************************************************************************
                  ;Interrupt Service Routine (ISR)
*****************************************************************************
                  ;
                  ;Interrupts occur when
                  ;
                  ;
              ISR: STORE s0, ISR_preserve_s0        ;preserve s0
                  ;
                  INPUT int_kind, interrupt_kind    ;
                  FETCH NET_para, RF_mode           ;teset the current RF_mode
                TEST NET_para, 01                   ;
                  JUMP NZ, Base_interrupt           ;
                  ;
                  ;
 End_interrupt: TEST int_kind, change_to_base       ;check and set change_mode flag

                  JUMP NZ, Change_to_base           ;
                  CALL Unicast_to_base              ;
                  JUMP end_ISR
                  ;
Base_interrupt: TEST int_kind, broad_to_end         ;
                  JUMP Z, Broad_to_base             ;
                  CALL Broad_to_end                 ;
                  ;
        end_ISR: INPUT sD, status                   ;CHECK if the tx has finished
                TEST    sD, RF_tx_state             ;
                JUMP    NZ, end_ISR                 ;
                LOAD    s0, 01                      ;if so, set the flag and output
                OUTPUT s0 ,TX_over_port             ;
                FETCH s0, ISR_preserve_s0           ;restore s0
                  RETURNI ENABLE
                  ;
*****************************************************************************
                  ;Interrupt Vector
*****************************************************************************
                  ADDRESS 3FF
                  JUMP ISR
```

lcd 时序控制代码

```
**************************************************************************
                    ;Port definitions
**************************************************************************
                    ;
                    CONSTANT status, 00                    ; status read port
                    CONSTANT UART_data_present, 08         ; uart rx data present --bit 3
                    CONSTANT rx_buffer_full, 02            ; uart rx buffer full    --bit 1
                    ;LCD interface ports
                    CONSTANT LCD_output_port, 10
                    CONSTANT LCD_E, 01
                    CONSTANT LCD_RW, 02
                    CONSTANT LCD_RS, 04
                    CONSTANT LCD_drive, 08
                    CONSTANT LCD_DB4, 10
                    CONSTANT LCD_DB5, 20
                    CONSTANT LCD_DB6, 40
                    CONSTANT LCD_DB7, 80

                    CONSTANT LCD_input_port, 20
                    CONSTANT LCD_read_spare0, 01
                    CONSTANT LCD_read_spare1, 02
                    CONSTANT LCD_read_spare2, 04
                    CONSTANT LCD_read_spare3, 08
                    CONSTANT LCD_read_DB4, 10
                    CONSTANT LCD_read_DB5, 20
                    CONSTANT LCD_read_DB6, 40
                    CONSTANT LCD_read_DB7, 80              ;
                    ;
                    CONSTANT interrupt_kind, 40            ; interrupt kind input
                    CONSTANT TPM_clear_ack,10              ; TPM_clear_ack
                     CONSTANT change_mode_flag,08          ;
                     CONSTANT LCD_clear_disp, 04           ;
                    CONSTANT LCD_shift_disp_left, 02       ;
                    CONSTANT LCD_shift_disp_right, 01      ;
**************************************************************************
                    ;Special Register usage
**************************************************************************
                    NAMEREG sF, UART_data                  ;used to pass data to and
from the UART
                    ;
**************************************************************************
                    ;Scratch Pad Memory Locations
                    ;*********************************************************
```

```
**************************
                CONSTANT ISR_preserve_s0, 00            ;Preserve s0 contents during
ISR
            CONSTANT RF_mode,           01          ; remember the RF operation
mode
                ;
*****************************************************************************
                ;Useful data constants
*****************************************************************************
                CONSTANT shift_delay_msb, 01            ;
                CONSTANT shift_delay_lsb, F4
            CONSTANT delay_1us_constant,01    ;10Mhz
*****************************************************************************
                ;Initialise the system
*****************************************************************************
                ;
    cold_start: CALL LCD_reset                          ;initialise LCD display
            LOAD sE, 00                                 ;
            STORE sE, RF_mode                           ;
                ENABLE INTERRUPT
                ;
                ;Write welcome message to LCD display
                ;
                LOAD s5, 10                             ;Line 1 position 0
                CALL LCD_cursor
                CALL disp_XIDIAN                        ;Display Contents
            CALL disp_END                               ;

                LOAD s5, 20
                CALL LCD_cursor
                ;
                ;wait for the data from rx and display them
                ;
  WAIT_for_rx: INPUT s1, status                         ; check status for data
                TEST s1, UART_data_present              ; used to set flags
                JUMP Z, WAIT_for_rx                     ; nothing ,loop again
                INPUT UART_data, UART_rx                ; input the rx data
                CALL disp_RX                            ; display the received data
                JUMP WAIT_for_rx                        ; wait for the next
                ;
                ;LCD_shift_delay:  LOAD  sF,  shift_delay_msb                ;
[sF,sE]=loop delay in ms
                ;LOAD sE, shift_delay_lsb
                ;LCD_delay_loop: CALL delay_1ms
```

```
                ;SUB sE, 01                     ;decrement delay counter
                ;SUBCY sF, 00
                ;JUMP NC, LCD_delay_loop
                ;CALL LCD_shift_left            ;shift LCD display
                ;JUMP LCD_shift_delay
                ;
*************************************************************************
                ;LCD text messages
*************************************************************************
                ;
                ;display UART_data on LCD at current cursor position
        disp_RX: LOAD s5, UART_data
                CALL LCD_write_data
                RETURN
                ;
                ;
                ;Display contents on LCD at current cursor position
                ;
     disp_XIDIAN: LOAD s5, character_X
                CALL LCD_write_data
                LOAD s5, character_I
                CALL LCD_write_data
                LOAD s5, character_D
                CALL LCD_write_data
                LOAD s5, character_I
                CALL LCD_write_data
                LOAD s5, character_A
                CALL LCD_write_data
                LOAD s5, character_N
                CALL LCD_write_data
                LOAD s5, character_COLON
                CALL LCD_write_data
                RETURN
                ;
                ;
    disp_END: LOAD s5, character_E
                CALL LCD_write_data
                LOAD s5, character_N
                CALL LCD_write_data
                LOAD s5, character_D
                CALL LCD_write_data
                LOAD s5, character_UNDER
                CALL LCD_write_data
                LOAD s5, character_M
```

```
                    CALL LCD_write_data
                    LOAD s5, character_0
                    CALL LCD_write_data
                    LOAD s5, character_D
                    CALL LCD_write_data
                    LOAD s5, character_E
                    CALL LCD_write_data
                    RETURN

disp_BASE: LOAD s5, character_B
                    CALL LCD_write_data
                    LOAD s5, character_A
                    CALL LCD_write_data
                    LOAD s5, character_S
                    CALL LCD_write_data
               LOAD s5, character_E
                    CALL LCD_write_data
                    LOAD s5, character_UNDER
                    CALL LCD_write_data
                    LOAD s5, character_M
                    CALL LCD_write_data
                    LOAD s5, character_0
                    CALL LCD_write_data
                    LOAD s5, character_D
                    CALL LCD_write_data
                    LOAD s5, character_E
                    CALL LCD_write_data
                    RETURN
                    ;Display a space on LCD at current cursor position
                    ;
   disp_space: LOAD s5, character_space
                    CALL LCD_write_data
                    RETURN
                    ;
******************************************************************************
                    ;Software delay routines
******************************************************************************
                    ;
                    ;Delay of 1us.
                    ;
    delay_1us: LOAD s0, delay_1us_constant
     wait_1us: SUB s0, 01
                    JUMP NZ, wait_1us
                    RETURN
```

```
                ;
                ;Delay of 40us.
                ;
                ;Registers used s0, s1
                ;
delay_40us: LOAD s1, 28                         ;40 x 1us = 40us
 wait_40us: CALL delay_1us
                SUB s1, 01
                JUMP NZ, wait_40us
                RETURN
                ;
                ;
                ;Delay of 1ms.
                ;
                ;Registers used s0, s1, s2
                ;
 delay_1ms: LOAD s2, 19                         ;25 x 40us = 1ms
  wait_1ms: CALL delay_40us
                SUB s2, 01
                JUMP NZ, wait_1ms
                RETURN
                ;
                ;Delay of 20ms.
                ;
                ;Delay of 20ms used during initialisation.
                ;
                ;Registers used s0, s1, s2, s3
                ;
delay_20ms: LOAD s3, 14                         ;20 x 1ms = 20ms
 wait_20ms: CALL delay_1ms
                SUB s3, 01
                JUMP NZ, wait_20ms
                RETURN
                ;
                ;Delay of approximately 1 second.
                ;
                ;Registers used s0, s1, s2, s3, s4
                ;
   delay_1s: LOAD s4, 32                        ;50 x 20ms = 1000ms
    wait_1s: CALL delay_20ms
                SUB s4, 01
                JUMP NZ, wait_1s
                RETURN
************************************************************
```

```
                    ;LCD Character Module Routines
**********************************************************
     LCD_pulse_E: XOR s4, LCD_E                    ;E=1
                    OUTPUT s4, LCD_output_port
                    CALL delay_1us
                    XOR s4, LCD_E                  ;E=0
                    OUTPUT s4, LCD_output_port
                    RETURN
                    ;
LCD_write_inst4: AND s4, F8                       ;Enable=1 RS=0 Instruction,
RW=0 Write, E=0
                    OUTPUT s4, LCD_output_port    ;set up RS and RW >40ns
before enable pulse
                    CALL LCD_pulse_E
                    RETURN
                    ;
LCD_write_inst8: LOAD s4, s5
                    AND s4, F0                    ;Enable=0 RS=0 Instruction,
RW=0 Write, E=0
                    OR s4, LCD_drive              ;Enable=1
                    CALL LCD_write_inst4          ;write upper nibble
                    CALL delay_1us                ;wait >1us
                    LOAD s4, s5                   ;select lower nibble with
                    SL1 s4                        ;Enable=1
                    SL0 s4                        ;RS=0 Instruction
                    SL0 s4                        ;RW=0 Write
                    SL0 s4                        ;E=0
                    CALL LCD_write_inst4          ;write lower nibble
                    CALL delay_40us               ;wait >40us
                    LOAD s4, F0                   ;Enable=0 RS=0 Instruction,
RW=0 Write, E=0
                    OUTPUT s4, LCD_output_port    ;Release master enable
                    RETURN
                    ;
 LCD_write_data: LOAD s4, s5
                    AND s4, F0                    ;Enable=0 RS=0 Instruction,
RW=0 Write, E=0
                    OR s4, 0C                     ;Enable=1 RS=1 Data,
RW=0 Write, E=0
                    OUTPUT s4, LCD_output_port    ;set up RS and RW >40ns
before enable pulse
                    CALL LCD_pulse_E              ;write upper nibble
                    CALL delay_1us                ;wait >1us
                    LOAD s4, s5                   ;select lower nibble with
```

```
                SL1 s4                          ;Enable=1
                SL1 s4                          ;RS=1 Data
                SL0 s4                          ;RW=0 Write
                SL0 s4                          ;E=0
                OUTPUT s4, LCD_output_port      ;set up RS and RW >40ns
before enable pulse
                CALL LCD_pulse_E                ;write lower nibble
                CALL delay_40us                 ;wait >40us
                LOAD s4, F0                     ;Enable=0 RS=0 Instruction,
RW=0 Write, E=0
                OUTPUT s4, LCD_output_port      ;Release master enable
                RETURN
                ;
                ;
 LCD_read_data8: LOAD s4, 0E                    ;
                OUTPUT s4, LCD_output_port      ;
                XOR s4, LCD_E                   ;E=1
                OUTPUT s4, LCD_output_port
                CALL delay_1us                  ;wait >260ns to access data
                INPUT s5, LCD_input_port        ;read upper nibble
                XOR s4, LCD_E                   ;E=0
                OUTPUT s4, LCD_output_port
                CALL delay_1us                  ;wait >1us
                XOR s4, LCD_E                   ;E=1
                OUTPUT s4, LCD_output_port
                CALL delay_1us                  ;wait >260ns to access data
                INPUT s0, LCD_input_port        ;read lower nibble
                XOR s4, LCD_E                   ;E=0
                OUTPUT s4, LCD_output_port
                AND s5, F0                      ;merge upper and lower
nibbles
                SR0 s0
                SR0 s0
                SR0 s0
                SR0 s0
                OR s5, s0
                LOAD s4, 04                     ;Enable=0 RS=1 Data,
RW=0 Write, E=0
                OUTPUT s4, LCD_output_port      ;Stop reading 5V device and
release master enable
                CALL delay_40us                 ;wait >40us
                RETURN
                ;
        LCD_reset: CALL delay_20ms              ;
```

```
                LOAD s4, 30
                CALL LCD_write_inst4            ;send '3'
                CALL delay_20ms                  ;wait >4.1ms
                CALL LCD_write_inst4            ;send '3'
                CALL delay_1ms                   ;wait >100us
                CALL LCD_write_inst4            ;send '3'
                CALL delay_40us                 ;wait >40us
                LOAD s4, 20
                CALL LCD_write_inst4            ;send '2'
                CALL delay_40us                 ;wait >40us
                LOAD s5, 28                     ;Function set
                CALL LCD_write_inst8
                LOAD s5, 06                     ;Entry mode
                CALL LCD_write_inst8
                LOAD s5, 0F
                CALL LCD_write_inst8
   LCD_clear: LOAD s5, 01                       ;Display clear
                CALL LCD_write_inst8
                CALL delay_1ms                   ;
                CALL delay_1ms
                RETURN
                ;
  LCD_cursor: TEST s5, 10                       ;test for line 1
                JUMP Z, set_line2
                AND s5, 0F                      ;
                OR s5, 80
                CALL LCD_write_inst8
                RETURN
   set_line2:    AND s5, 0F
                OR s5, C0
                CALL LCD_write_inst8            ;
                RETURN
                ;
                ;This routine will shift the complete display one position to the left or
                ;
 LCD_shift_left: LOAD s5, 18                    ;shift display left
                CALL LCD_write_inst8
                RETURN
                ;
                ;
LCD_shift_right: LOAD s5, 1C                    ;shift display right
                CALL LCD_write_inst8
                RETURN
                ;
```

```
change_mode: LOAD sE, 01                    ;
             STORE sE, RF_mode              ;
             LOAD    s5, 17                 ;
             CALL    LCD_cursor             ;
             CALL    disp_BASE              ;
             LOAD s5,20                     ;
             CALL LCD_cursor                ;
             RETURN                         ;
LCD_clear_and_dis:LOAD s5, 01                    ;Display clear
             CALL LCD_write_inst8
             CALL delay_1ms                 ;
             CALL delay_1ms                 ;
              LOAD s5, 10                   ;
             CALL    LCD_cursor             ;
             CALL disp_XIDIAN
             FETCH sE, RF_mode              ;
             TEST    sE,    01              ;
             CALL    Z, disp_END            ;
             CALL NZ, disp_BASE             ;
             LOAD s5, 20                    ;
             CALL LCD_cursor                ;
             RETURN

dis_clear_ack:CALL LCD_clear_and_dis         ;
             LOAD s5, character_A
             CALL LCD_write_data
             LOAD s5, character_T
             CALL LCD_write_data
             LOAD s5, character_T
             CALL LCD_write_data
             LOAD s5, character_A
             CALL LCD_write_data
             LOAD s5, character_C
             CALL LCD_write_data
             LOAD s5, character_K
             CALL LCD_write_data
             LOAD s5, character_E
             CALL LCD_write_data
             LOAD s5, character_D
             CALL LCD_write_data
             LOAD s5, character_comma
             CALL LCD_write_data
             LOAD s5, character_K
             CALL LCD_write_data
```

```
                LOAD s5, character_E
                CALL LCD_write_data
                LOAD s5, character_Y
                CALL LCD_write_data
                LOAD s5, character_space
                CALL LCD_write_data
                LOAD s5, character_C
                CALL LCD_write_data
                LOAD s5, character_L
                CALL LCD_write_data
                LOAD s5, character_E
                CALL LCD_write_data
                LOAD s5, character_A
                CALL LCD_write_data
                LOAD s5, character_R
                CALL LCD_write_data
                LOAD s5, character_E
                CALL LCD_write_data
                LOAD s5, character_D
                CALL LCD_write_data
                RETURN
*****************************************************************************
                ;Interrupt Service Routine (ISR)
*****************************************************************************
          ISR: STORE s0, ISR_preserve_s0
                INPUT s0, interrupt_kind
                TEST s0, LCD_clear_disp
                CALL NZ, LCD_clear_and_dis
                TEST s0, LCD_shift_disp_left
                CALL NZ, LCD_shift_left
                TEST s0, LCD_shift_disp_right
                CALL NZ, LCD_shift_right
               TEST s0, change_mode_flag
               CALL NZ, change_mode
               TEST s0, TPM_clear_ack
               CALL NZ, dis_clear_ack
      end_ISR: FETCH s0, ISR_preserve_s0          ;restore s0
                RETURNI ENABLE
*****************************************************************************
                ;Interrupt Vector
*****************************************************************************
                ;
                ADDRESS 3FF
                JUMP ISR
```

ps2 接收控制代码

```
;***************************************************************
;                          Description
;***************************************************************
;this code is intened for ps2_outfifo1vs4 program
;
;***************************************************************
;                       Port address definitions...
;***************************************************************
CONSTANT status, 00                 ; status read port
CONSTANT uart_data_present, 08 ; uart rx data present --bit 3
CONSTANT ps2_ascii_present, 04 ; ps2    receive data present    bit--2
CONSTANT rx_buffer_full, 02        ; uart rx buffer full    --bit 1
CONSTANT ps2_buffer_full, 01       ; ps2 receive buffer full --bit 0
;
;

CONSTANT LCD_data_tx, 02            ; LCD transmit write port
CONSTANT uart_data_rx, 08           ; UART receive read port
CONSTANT ps2_ascii, 04              ; ps2    ascii read port
CONSTANT out_fifo1vs8, 10           ;outfifo1vs8 write port
;
CONSTANT OVER_port, 01              ;
;
CONSTANT lcd_and_outfifo, 12
*****************************************************************************
                          Special Register usage
*****************************************************************************
NAMEREG sF, PS2_data                ;
NAMEREG s0, count_reg
*****************************************************************************
;                                          Scratch Pad Memory Locations
*****************************************************************************
CONSTANT FIFO_data_counter, 00 ;fifo1vs8 counter for 4 datas once a
  time to transmit
;***************************************************************
cold_start: LOAD count_reg, all_clear                ; zero out reg count_reg
LOAD count_reg, 04                      ;
STORE count_reg, FIFO_data_counter      ;initialize couter=4
*****************************************************************
;Main program
*****************************************************************
;Description: wait for the ps2_port data and write it back to uart_tx and
outfifo,
```

```
                ;if the messgae ended with ascii_CR is shorter than 4,make it for 4
                ;
                ;
                ;Check ps2_port for data and write it back to uart_tx and outfifo1vs8
                ;
    ps2_echo: INPUT s1, status                  ; check status for data
                TEST s1, ps2_ascii_present      ; used to set flags
                JUMP Z, ps2_echo                ; nothing, loop again
                INPUT PS2_data, ps2_ascii               ; read received byte
                OUTPUT PS2_data, lcd_and_outfifo        ; write it back out
                FETCH count_reg, FIFO_data_counter  ; SUB the fifo1vs8 counter 1
                SUB count_reg, ONE                  ;
                CALL Z, RESET_counter               ; if counter=0, then JUMP

                STORE count_reg, FIFO_data_counter
                COMPARE PS2_data, ascii_CR          ;
                CALL Z, MAKE_for_4
                JUMP ps2_echo                       ; loop again
RESET_counter: LOAD count_reg, 04               ;
                RETURN
                ;
  MAKE_for_4: FETCH count_reg, FIFO_data_counter    ;
                COMPARE count_reg, 04               ;
                JUMP Z, out_end_info                ;
                LOAD s1, ascii_SPACE            ;
                OUTPUT s1, lcd_and_outfifo      ;
                FETCH count_reg, FIFO_data_counter  ; SUB the fifo1vs8 counter 1
                SUB count_reg, ONE                  ;
                CALL Z, RESET_counter
                STORE count_reg, FIFO_data_counter
                JUMP MAKE_for_4
                RETURN                              ;
  out_end_info: LOAD s2, 01                         ;
                OUTPUT s2, OVER_port                ;
                RETURN                              ;
```