```vhdl
------------------------------------------------------------------------
-- HEIG-VD, Haute Ecole d'Ingenierie et de Gestion du canton de Vaud
-- Institut REDS, Reconfigurable & Embedded Digital Systems
--
-- File         : axi4lite_slave.vhd
-- Author       : E. Messerli    27.07.2017
-- Description  : slave interface AXI  (without burst)
-- used for     : SOCF lab
--| Modifications |-----------------------------------------------------------
-- Ver  Date       Auteur  Description
-- 1.0  26.03.2019  EMI    Adaptation du chablon pour les etudiants
-- 1.1  03.04.2020  ISS    Complète le chablon pour le laboratoire 5
------------------------------------------------------------------------------

library ieee;
    use ieee.std_logic_1164.all;
    use ieee.numeric_std.all;

entity axi4lite_slave is
    generic (
        -- Users to add parameters here

        -- User parameters ends

        -- Width of S_AXI data bus
        AXI_DATA_WIDTH  : integer   := 32;  -- 32 or 64 bits
        -- Width of S_AXI address bus
        AXI_ADDR_WIDTH  : integer   := 12
    );
    port (
        -- AXI4-Lite
        axi_clk_i       : in  std_logic;
        axi_reset_i     : in  std_logic;

        -- Write Address Channel
        axi_awaddr_i    : in  std_logic_vector(AXI_ADDR_WIDTH-1 downto 0);
        axi_awprot_i    : in  std_logic_vector( 2 downto 0);    -- not used
        axi_awvalid_i   : in  std_logic;
        axi_awready_o   : out std_logic;

        -- Write Data Channel
        axi_wdata_i     : in  std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
        axi_wstrb_i     : in std_logic_vector((AXI_DATA_WIDTH/8)-1 downto 0);
        axi_wvalid_i    : in  std_logic;
        axi_wready_o    : out std_logic;

        -- Write Response Channel
        axi_bresp_o     : out std_logic_vector(1 downto 0);
        axi_bvalid_o    : out std_logic;
        axi_bready_i    : in  std_logic;

        -- Read Address Channel
        axi_araddr_i    : in  std_logic_vector(AXI_ADDR_WIDTH-1 downto 0);
        axi_arprot_i    : in  std_logic_vector( 2 downto 0);    -- not used
        axi_arvalid_i   : in  std_logic;
        axi_arready_o   : out std_logic;

        -- Read Data Channel
        axi_rdata_o     : out std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
        axi_rresp_o     : out std_logic_vector(1 downto 0);
        axi_rvalid_o    : out std_logic;
        axi_rready_i    : in  std_logic;

        -- User input-output
        switch_i        : in  std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
        key_i           : in  std_logic_vector(AXI_DATA_WIDTH-1 downto 0);

        leds_o          : out std_logic_vector(AXI_DATA_WIDTH-1 downto 0);

        hex0_o          : out std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
        hex1_o          : out std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
        hex2_o          : out std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
        hex3_o          : out std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
        hex4_o          : out std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
        hex5_o          : out std_logic_vector(AXI_DATA_WIDTH-1 downto 0);


        -- Interruption
        irq_o           : out std_logic
    );
end entity axi4lite_slave;

architecture rtl of axi4lite_slave is

    signal reset_s : std_logic;

    -- local parameter for addressing 32 bit / 64 bits, cst: AXI_DATA_WIDTH
    -- ADDR_LSB is used for addressing word 32/64 bits registers/memories
    -- ADDR_LSB = 2 for 32 bits (n-1 downto 2)
    -- ADDR_LSB = 3 for 64 bits (n-1 downto 3)
    constant ADDR_LSB           : integer := (AXI_DATA_WIDTH/32)+ 1;

    --------------- SIGNAUX AXI 4 LIGHT -------------------

    --signal for the AXI slave
    --intern signal for output
    signal axi_awready_s        : std_logic;
    signal axi_arready_s        : std_logic;

    signal axi_wready_s         : std_logic;
    signal axi_rready_s         : std_logic;

    signal axi_rvalid_s         : std_logic;
    signal axi_rresp_s          : std_logic_vector(1 downto 0);
    signal axi_rdata_mem_s      : std_logic_vector(AXI_DATA_WIDTH-1 downto 0);

    -- write enable
    signal axi_data_wren_s      : std_logic;

     --intern signal for the axi interface
    signal axi_waddr_mem_s      : std_logic_vector(AXI_ADDR_WIDTH-1 downto ADDR_LSB);
    signal axi_araddr_mem_s     : std_logic_vector(AXI_ADDR_WIDTH-1 downto ADDR_LSB);

    signal axi_wdata_mem_s      : std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
    signal axi_wstrb_mem_s      : std_logic_vector((AXI_DATA_WIDTH/8)-1 downto 0);
    -- signal axi_araddr_mem_s    : std_logic_vector(AXI_ADDR_WIDTH-1 downto ADDR_LSB);

    signal axi_bresp_s          : std_logic_vector(1 downto 0);
    signal axi_bvalid_s         : std_logic;


    --------------- SIGNAUX ENTREES / SORTIES ---------------

    constant registre_cst_mem   : std_logic_vector(AXI_DATA_WIDTH-1 downto 0):=
    x"deedbeef";
    signal registre_test_mem    : std_logic_vector(AXI_DATA_WIDTH-1 downto 0):=
    x"12345678";

    -- signal for registre input (switch / key)
    signal registre_switch_mem  : std_logic_vector(9 downto 0) := (others => 'X');
    signal registre_key_mem     : std_logic_vector(3 downto 0) := (others => 'X');

    -- signal for registre leds
    signal registre_led_mem     : std_logic_vector(9 downto 0) := (others => 'X');

    -- signal for registre 7 seg
    signal registre_hex0_mem    : std_logic_vector(6 downto 0) := (others => 'X');
    signal registre_hex1_mem    : std_logic_vector(6 downto 0) := (others => 'X');
    signal registre_hex2_mem    : std_logic_vector(6 downto 0) := (others => 'X');
    signal registre_hex3_mem    : std_logic_vector(6 downto 0) := (others => 'X');
    signal registre_hex4_mem    : std_logic_vector(6 downto 0) := (others => 'X');
    signal registre_hex5_mem    : std_logic_vector(6 downto 0) := (others => 'X');

    --------------- SIGNAUX GESTION IRQ ---------------
    signal irq_s                : std_logic;
    signal irq_source           : std_logic_vector(3 downto 0) := (others => '0');
    signal key_val_save         : std_logic_vector(3 downto 0) := (others => '1');
    -- par défaut, toutes les irq actives
    signal key_irq_mask         : std_logic_vector(3 downto 0) := (others => '0');

begin

     -- mise à jour des entrées
    reset_s  <= axi_reset_i;

    registre_switch_mem <= switch_i(9 downto 0);
    registre_key_mem    <= key_i(3 downto 0);



-----------------------------------------------------------
-- Write address channel

    process (reset_s, axi_clk_i)
    begin
        -- En cas de reset
        if reset_s = '1' then
            -- Valeur par défaut
            axi_awready_s <= '0';
            axi_waddr_mem_s <= (others => '0');
        elsif rising_edge(axi_clk_i) then
            -- Si une adresse d'écriture est valide
            if (axi_awready_s = '0' and axi_awvalid_i = '1')  then --and axi_wvalid_i = '1') then  modif EMI 10juil2018
                -- slave is ready to accept write address when
                -- there is a valid write address
                axi_awready_s <= '1';
                -- Write Address memorizing
                axi_waddr_mem_s <= axi_awaddr_i(AXI_ADDR_WIDTH-1 downto ADDR_LSB);
            else
                axi_awready_s <= '0';
                axi_waddr_mem_s <= (others => '0');
            end if;
        end if;
    end process;
    axi_awready_o <= axi_awready_s;


-----------------------------------------------------------
-- Write data channel

    -- Implement axi_wready generation
    process (reset_s, axi_clk_i)
    begin
        -- En cas de reset
        if reset_s = '1' then
            -- Valeur par défaut
            axi_wready_s    <= '0';
            axi_wdata_mem_s <= (others => '0');
            axi_wstrb_mem_s <= (others => '0');
        elsif rising_edge(axi_clk_i) then
            -- Si les données d'écriture est valide
            if (axi_wready_s = '0' and axi_wvalid_i = '1')  then
                -- slave is ready to accept write data when
                -- there is a valid write data
                axi_wready_s <= '1';

                -- Read axi_wstrb_i
                axi_wstrb_mem_s <= axi_wstrb_i((AXI_DATA_WIDTH/8)-1 downto 0);


                -- Mémorisation des données à écrire en fonction du paramètre strobe
                axi_wdata_mem_s <= (others => '0');

                if (axi_wstrb_i(0) = '1') then
                    axi_wdata_mem_s(7 downto 0) <= axi_wdata_i(7 downto 0);
                end if;
                if (axi_wstrb_i(1) = '1') then
                    axi_wdata_mem_s(15 downto 8) <= axi_wdata_i(15 downto 8);
                end if;
                if (axi_wstrb_i(2) = '1') then
                    axi_wdata_mem_s(23 downto 16) <= axi_wdata_i(23 downto 16);
                end if;
                if (axi_wstrb_i(3) = '1') then
                    axi_wdata_mem_s(31 downto 24) <= axi_wdata_i(31 downto 24);
                end if;

                -- Test sans la fonctionnalité strobe
                -- axi_wdata_mem_s <= axi_wdata_i;

            else
                axi_wready_s <= '0';
                axi_wdata_mem_s <= (others => '0');
                axi_wstrb_mem_s <= (others => '0');

            end if;
        end if;
    end process;

    -- Met à jour la sortie
    axi_wready_o <= axi_wready_s;


    -- condition to write data : si on est prêt à écrire
    axi_data_wren_s <= '1' when axi_wready_s = '1' else
                       '0';


    process (reset_s, axi_clk_i)
        --number address to access 32 or 64 bits data
        variable int_waddr_v : natural;
    begin
        if reset_s = '1' then
            -- Valeur par défaut : RESET
            registre_test_mem <= x"12345678";
            registre_led_mem  <= "0101010101";
            registre_hex0_mem <= "1000000" ;
            registre_hex1_mem <= "1111001";
            registre_hex2_mem <= "0100100";
            registre_hex3_mem <= "0110000";
            registre_hex4_mem <= "0011001";
            registre_hex5_mem <= "0010010";

            key_irq_mask      <= "0000";

        elsif rising_edge(axi_clk_i) then
            -- Si une écriture est active
            if axi_data_wren_s = '1' then
                -- convertie l'adresse d'écriture en integer
                int_waddr_v   := to_integer(unsigned(axi_waddr_mem_s));
                case int_waddr_v is
                    -- offset 0 : constante
                    when 0    =>
                    -- offset 4 : registre de test
                    when 1    =>
                        registre_test_mem <= axi_wdata_mem_s;

                    -- offset 64 : leds
                    when 64   =>
                        registre_led_mem <= axi_wdata_mem_s(9 downto 0);

                    -- offset 130 : mask irq key
                    when 130   =>
                        key_irq_mask <= axi_wdata_mem_s(3 downto 0);

                    -- offset 256 - 276 : afficheur 7 seg
                    when 256   =>
                        registre_hex0_mem <= axi_wdata_mem_s(6 downto 0);
                    when 260   =>
                        registre_hex1_mem <= axi_wdata_mem_s(6 downto 0);
                    when 264   =>
                        registre_hex2_mem <= axi_wdata_mem_s(6 downto 0);
                    when 268   =>
                        registre_hex3_mem <= axi_wdata_mem_s(6 downto 0);
                    when 272   =>
                        registre_hex4_mem <= axi_wdata_mem_s(6 downto 0);
                    when 276   =>
                        registre_hex5_mem <= axi_wdata_mem_s(6 downto 0);


                    when others => null;
                end case;
            end if;
        end if;
    end process;


-----------------------------------------------------------
-- Write response channel

    process (reset_s, axi_clk_i)
    begin
        -- En cas de reset
        if reset_s = '1' then
            -- Valeur par défaut
            axi_bresp_s    <= "00";
            axi_bvalid_s   <= '0';
        elsif rising_edge(axi_clk_i) then
            -- Si le master est pret à lire la réponse
            if (axi_bvalid_s = '0' and axi_bready_i = '1')  then
                -- slave is ready to accept write data when
                -- there is a valid write data
                axi_bvalid_s <= '1';
                -- Write response
                axi_bresp_s    <= "00";
            else
                axi_bvalid_s <= '0';
                axi_bresp_s    <= "--";

            end if;
        end if;
    end process;
    -- Met à jours les sorties
    axi_bresp_o <= axi_bresp_s;
    axi_bvalid_o <= axi_bvalid_s;



-----------------------------------------------------------
-- Read address channel

    process (reset_s, axi_clk_i)
    begin
        -- en cas de reset
        if reset_s = '1' then
```

```
            -- valeur par défaut
           axi_arready_s    <= '0';
           axi_araddr_mem_s <= (others => '1');
        elsif rising_edge(axi_clk_i) then
            -- Si une adresse de lecture est valide
            if axi_arready_s = '0' and axi_arvalid_i = '1' then
                -- indicates that the slave has acceped the valid read address
                axi_arready_s    <= '1';
                -- Read Address memorizing
                axi_araddr_mem_s <= axi_araddr_i(AXI_ADDR_WIDTH-1 downto ADDR_LSB);
            else
                axi_arready_s    <= '0';
            end if;
        end if;
    end process;
    -- Met à jour la sortie
    axi_arready_o <= axi_arready_s;

-----------------------------------------------------------
-- Read data channel

    -- Implement axi_wready generation
    process (reset_s, axi_clk_i)
    --number address to access 32 or 64 bits data
        variable int_raddr_v : natural;
    begin

        -- En cas de reset
        if reset_s = '1' then
            -- valeur par défaut
            axi_rvalid_s    <= '0';
            axi_rdata_mem_s <= (others => '0');
            axi_rresp_s    <= "00";

            irq_source <= "0000";
            irq_s <= '0';

        elsif rising_edge(axi_clk_i) then
            -- Gestion des interruptions
            if (key_val_save(0) /= registre_key_mem(0) and registre_key_mem(0) = '0' and key_irq_mask(0) = '0') then
                irq_source(0) <= '1';
                irq_s <= '1';
            elsif (key_val_save(1) /= registre_key_mem(1) and registre_key_mem(1) = '0' and key_irq_mask(1) = '0') then
                irq_source(1) <= '1';
                irq_s <= '1';
            elsif (key_val_save(2) /= registre_key_mem(2) and registre_key_mem(2) = '0' and key_irq_mask(2) = '0') then
                irq_source(2) <= '1';
                irq_s <= '1';
            elsif (key_val_save(3) /= registre_key_mem(3) and registre_key_mem(3) = '0' and key_irq_mask(3) = '0') then
                irq_source(3) <= '1';
                irq_s <= '1';
            end if;
            -- Met à jour l'ancienne valeur des keys
            key_val_save <= registre_key_mem;

            -- Si une lecture est faite
            if (axi_arready_s = '1' and axi_rvalid_s = '0')  then

                -- Pré-charge une lecture à 0
                axi_rdata_mem_s <= (others => '0');

                -- slave is ready to accept write data when
                -- there is a valid write data
                axi_rvalid_s <= '1';

```

```
                    -- read Data go
                    int_raddr_v   := to_integer(unsigned(axi_araddr_mem_s));
                    axi_rresp_s    <= "00";

                    -- En fonction de l'adresse qu'on souhaite lire
                    case int_raddr_v is
                        -- Lecture de la constante
                        when 0    =>
                            axi_rdata_mem_s <= registre_cst_mem;
                        -- Lecture du registre de test
                        when 1    =>
                            axi_rdata_mem_s <= registre_test_mem;
                        -- Lecture des leds
                        when 64    =>
                            axi_rdata_mem_s(9 downto 0) <= registre_led_mem;
                         -- Lecture des keys
                        when 128    =>
                            axi_rdata_mem_s(3 downto 0) <= registre_key_mem;
                        -- lecture de la source d'interruption et acquitement
                        when 129    =>
                            axi_rdata_mem_s(3 downto 0) <= irq_source;
                            irq_s <= '0';
                            irq_source <= "0000";

                        -- lecture des masque des irq
                        when 130    =>
                            axi_rdata_mem_s(3 downto 0) <= key_irq_mask;
                        -- Lecture des switches
                        when 192    =>
                            axi_rdata_mem_s(9 downto 0) <= registre_switch_mem;

                        -- Lecture d'un afficheur 7 seg (256 - 276)
                        when 256    =>
                            axi_rdata_mem_s(6 downto 0) <= registre_hex0_mem;
                        when 260    =>
                            axi_rdata_mem_s(6 downto 0) <= registre_hex1_mem;
                        when 264    =>
                            axi_rdata_mem_s(6 downto 0) <= registre_hex2_mem;
                        when 268    =>
                            axi_rdata_mem_s(6 downto 0) <= registre_hex3_mem;
                        when 272    =>
                            axi_rdata_mem_s(6 downto 0) <= registre_hex4_mem;
                        when 276    =>
                            axi_rdata_mem_s(6 downto 0) <= registre_hex5_mem;


                        when others =>
                            axi_rresp_s    <= "00";
                    end case;

                else
                    axi_rvalid_s <= '0';
                    axi_rresp_s    <= "--";

                end if;
            end if;
        end process;

        -- Mise à jour de la ligne l'interruption
        irq_o <= irq_s;

        -- Mise à jour de la validité de lecture
        axi_rvalid_o <= axi_rvalid_s;

        -- Mise à jour des données lues
        axi_rdata_o <= axi_rdata_mem_s;

        -- Mise à jour de la réponse de lecture
        axi_rresp_o <= axi_rresp_s;
```

```vhdl


    -- Mise à jour des sorties
    leds_o(9 downto 0)     <=  registre_led_mem;

    hex0_o(6 downto 0)     <=  registre_hex0_mem;
    hex1_o(6 downto 0)     <=  registre_hex1_mem;
    hex2_o(6 downto 0)     <=  registre_hex2_mem;
    hex3_o(6 downto 0)     <=  registre_hex3_mem;
    hex4_o(6 downto 0)     <=  registre_hex4_mem;
    hex5_o(6 downto 0)     <=  registre_hex5_mem;


end rtl;

```

```vhdl
------------------------------------------------------------------------------------------
-- HEIG-VD
-- Haute Ecole d'Ingenerie et de Gestion du Canton de Vaud
-- School of Business and Engineering in Canton de Vaud
------------------------------------------------------------------------------------------
-- REDS Institute
-- Reconfigurable Embedded Digital Systems
------------------------------------------------------------------------------------------
--
-- File                 : DE1_SoC_top.vhd
-- Author               : Sébastien Masle
-- Date                 : 17.01.2018
--
-- Context              : HPA
--
------------------------------------------------------------------------------------------
-- Description : top design for DE1-SoC board
--
------------------------------------------------------------------------------------------
-- Dependencies :
--
------------------------------------------------------------------------------------------
-- Modifications :
-- Ver    Date        Engineer      Comments
-- 0.0    17.01.2018  SMS           Initial version.
--
------------------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity DE1_SoC_top is
    port ( -- clock pins
           CLOCK_50_i  : in  std_logic;
           CLOCK2_50_i : in  std_logic;
           CLOCK3_50_i : in  std_logic;
           CLOCK4_50_i : in  std_logic;

           -- ADC
           ADC_CS_N_o : out std_logic;
           ADC_DIN_o  : out std_logic;
           ADC_DOUT_i : in std_logic;
           ADC_SCLK_o : out std_logic;

           -- Audio
           AUD_ADCLRCK_io : inout std_logic;
           AUD_ADCDAT_i   : in std_logic;
           AUD_DACLRCK_io : inout std_logic;
           AUD_DACDAT_o   : out std_logic;
           AUD_XCK_o      : out std_logic;
           AUD_BCLK_io    : inout std_logic;

           -- SDRAM
           DRAM_ADDR_o  : out std_logic_vector(12 downto 0);
           DRAM_BA_o    : out std_logic_vector(1 downto 0);
           DRAM_CAS_N_o : out std_logic;
           DRAM_CKE_o   : out std_logic;
           DRAM_CLK_o   : out std_logic;
           DRAM_CS_N_o  : out std_logic;
           DRAM_DQ_io   : inout std_logic_vector(15 downto 0);
           DRAM_LDQM_o  : out std_logic;
           DRAM_RAS_N_o : out std_logic;
           DRAM_UDQM_o  : out std_logic;
```

```vhdl
        DRAM_WE_N_o  : out std_logic;

        --I2C Bus for Configuration of the Audio and Video-In Chips
        FPGA_I2C_SCLK_o  : out std_logic;
        FPGA_I2C_SDAT_io : inout std_logic;

        -- 40-pin headers
        GPIO_0_io    : inout std_logic_vector(35 downto 0);
        GPIO_1_io    : inout std_logic_vector(35 downto 0);

        -- Seven Segment Displays
        HEX0_o       : out std_logic_vector(6 downto 0);
        HEX1_o       : out std_logic_vector(6 downto 0);
        HEX2_o       : out std_logic_vector(6 downto 0);
        HEX3_o       : out std_logic_vector(6 downto 0);
        HEX4_o       : out std_logic_vector(6 downto 0);
        HEX5_o       : out std_logic_vector(6 downto 0);

        -- IR
        IRDA_RXD_i   : in std_logic;
        IRDA_TXD_o   : out std_logic;

        -- Pushbuttons
        KEY_i        : in std_logic_vector(3 downto 0);

        -- LEDs
        LEDR_o       : out std_logic_vector(9 downto 0);

        -- PS2 Ports
        PS2_CLK_io   : inout std_logic;
        PS2_DAT_io   : inout std_logic;
        PS2_CLK2_io  : inout std_logic;
        PS2_DAT2_io  : inout std_logic;

        -- Slider Switches
        SW_i         : in std_logic_vector(9 downto 0);

        -- Video-In
        TD_CLK27_i   : in std_logic;
        TD_DATA_i    : in std_logic_vector(7 downto 0);
        TD_HS_i      : in std_logic;
        TD_RESET_N_o : out std_logic;
        TD_VS_i      : in std_logic;

        -- VGA
        VGA_R_o       : out std_logic_vector(7 downto 0);
        VGA_G_o       : out std_logic_vector(7 downto 0);
        VGA_B_o       : out std_logic_vector(7 downto 0);
        VGA_CLK_o     : out std_logic;
        VGA_SYNC_N_o  : out std_logic;
        VGA_BLANK_N_o : out std_logic;
        VGA_HS_o      : out std_logic;
        VGA_VS_o      : out std_logic;

        -- DDR3 SDRAM
        HPS_DDR3_ADDR_o      : out std_logic_vector(14 downto 0);
        HPS_DDR3_BA_o        : out std_logic_vector(2 downto 0);
        HPS_DDR3_CAS_N_o     : out std_logic;
        HPS_DDR3_CKE_o       : out std_logic;
        HPS_DDR3_CK_N_o      : out std_logic;
        HPS_DDR3_CK_P_o      : out std_logic;
        HPS_DDR3_CS_N_o      : out std_logic;
        HPS_DDR3_DM_o        : out std_logic_vector(3 downto 0);
        HPS_DDR3_DQ_io       : inout std_logic_vector(31 downto 0);
        HPS_DDR3_DQS_N_io    : inout std_logic_vector(3 downto 0);
        HPS_DDR3_DQS_P_io    : inout std_logic_vector(3 downto 0);
        HPS_DDR3_ODT_o       : out std_logic;
        HPS_DDR3_RAS_N_o     : out std_logic;
        HPS_DDR3_RESET_N_o   : out std_logic;
```

```
            HPS_DDR3_RZQ_i      : in std_logic;
            HPS_DDR3_WE_N_o     : out std_logic;

            -- Ethernet
            --HPS_ENET_GTX_CLK_o   : out std_logic;
            --HPS_ENET_INT_N_io    : inout std_logic;
            --HPS_ENET_MDC_o       : out std_logic;
            --HPS_ENET_MDIO_io     : inout std_logic;
            --HPS_ENET_RX_CLK_i    : in std_logic;
            --HPS_ENET_RX_DATA_i   : in std_logic_vector(3 downto 0);
            --HPS_ENET_RX_DV_i     : in std_logic;
            --HPS_ENET_TX_DATA_o   : out std_logic_vector(3 downto 0);
            --HPS_ENET_TX_EN_o     : out std_logic;

            -- Flash
            --HPS_FLASH_DATA_io    : inout std_logic_vector(3 downto 0);
            --HPS_FLASH_DCLK_o     : out std_logic;
            --HPS_FLASH_NCSO_o     : out std_logic;

            -- Accelerometer
            --HPS_GSENSOR_INT_io   : inout std_logic;

            -- General Purpose I/O
            --HPS_GPIO_io          : inout std_logic_vector(1 downto 0);

            -- I2C
            --HPS_I2C_CONTROL_io   : inout std_logic;
            --HPS_I2C1_SCLK_io     : inout std_logic;
            --HPS_I2C1_SDAT_io     : inout std_logic;
            --HPS_I2C2_SCLK_io     : inout std_logic;
            --HPS_I2C2_SDAT_io     : inout std_logic;

            -- Pushbutton
            HPS_KEY_io          : inout std_logic;

            -- LED
            HPS_LED_io          : inout std_logic;

            -- SD Card
            --HPS_SD_CLK_o         : out std_logic;
            --HPS_SD_CMD_io        : inout std_logic;
            --HPS_SD_DATA_io       : inout std_logic_vector(3 downto 0);

            -- SPI
            --HPS_SPIM_CLK_o       : out std_logic;
            --HPS_SPIM_MISO_i      : in std_logic;
            --HPS_SPIM_MOSI_o      : out std_logic;
            --HPS_SPIM_SS_io       : inout std_logic;

            -- UART
            --HPS_UART_RX_i        : in std_logic;
            --HPS_UART_TX_o        : out std_logic;

            -- USB
            --HPS_CONV_USB_N_io    : inout std_logic;
            --HPS_USB_CLKOUT_i     : in std_logic;
            --HPS_USB_DATA_io      : inout std_logic_vector(7 downto 0);
            --HPS_USB_DIR_i        : in std_logic;
            --HPS_USB_NXT_i        : in std_logic;
            --HPS_USB_STP_o        : out std_logic;

            -- LTC connector
            --HPS_LTC_GPIO_io      : inout std_logic;

            -- FAN
            FAN_CTRL_o          : out std_logic
            );
    end DE1_SoC_top;

```

```vhdl
architecture top of DE1_SoC_top is

    component qsys_system is
        port (
            ------------------------------------
            -- FPGA Side
            ------------------------------------

            -- Global signals
            clk_clk                         : in    std_logic                       := 'X';             -- clk

            ------------------------------------
            -- HPS Side
            ------------------------------------
            -- DDR3 SDRAM
            memory_mem_a                    : out   std_logic_vector(14 downto 0);                      -- mem_a
            memory_mem_ba                   : out   std_logic_vector(2 downto 0);                       -- mem_ba
            memory_mem_ck                   : out   std_logic;                                          -- mem_ck
            memory_mem_ck_n                 : out   std_logic;                                          -- mem_ck_n
            memory_mem_cke                  : out   std_logic;                                          -- mem_cke
            memory_mem_cs_n                 : out   std_logic;                                          -- mem_cs_n
            memory_mem_ras_n                : out   std_logic;                                          -- mem_ras_n
            memory_mem_cas_n                : out   std_logic;                                          -- mem_cas_n
            memory_mem_we_n                 : out   std_logic;                                          -- mem_we_n
            memory_mem_reset_n              : out   std_logic;                                          -- mem_reset_n
            memory_mem_dq                   : inout std_logic_vector(31 downto 0) := (others => 'X'); -- mem_dq
            memory_mem_dqs                  : inout std_logic_vector(3 downto 0)  := (others => 'X'); -- mem_dqs
            memory_mem_dqs_n                : inout std_logic_vector(3 downto 0)  := (others => 'X'); -- mem_dqs_n
            memory_mem_odt                  : out   std_logic;                                          -- mem_odt
            memory_mem_dm                   : out   std_logic_vector(3 downto 0);                       -- mem_dm
            memory_oct_rzqin                : in    std_logic                       := 'X';             -- oct_rzqin

                conduit_export_switch_i             : in    std_logic_vector(31 downto 0) := (others => 'X'); -- switch_i
                conduit_export_key_i                : in    std_logic_vector(31 downto 0) := (others => 'X'); -- key_i

                conduit_export_leds_o               : out   std_logic_vector(31 downto 0);                      -- leds_o

                conduit_export_hex0_o               : out   std_logic_vector(31 downto 0);                      -- hex0_o
                conduit_export_hex1_o               : out   std_logic_vector(31 downto 0);                      -- hex1_o
                conduit_export_hex2_o               : out   std_logic_vector(31 downto 0);                      -- hex2_o
                conduit_export_hex3_o               : out   std_logic_vector(31 downto 0);                      -- hex3_o
                conduit_export_hex4_o               : out   std_logic_vector(31 downto 0);                      -- hex4_o
                conduit_export_hex5_o               : out   std_logic_vector(31 downto 0);                      -- hex5_o

            -- Pushbutton
            hps_io_hps_io_gpio_inst_GPIO54  : inout std_logic                    := 'X';             -- hps_io_gpio_inst_GPIO54

            -- LED
            hps_io_hps_io_gpio_inst_GPIO53  : inout std_logic                    := 'X'              -- hps_io_gpio_inst_GPIO53
        );
    end component qsys_system;

begin

---------------------------------------------------------
--  HPS mapping
---------------------------------------------------------

    System : component qsys_system
    port map (
        ------------------------------------
        -- FPGA Side
        ------------------------------------

            -- Global signals
            clk_clk             => CLOCK_50_i,

            ------------------------------------
            -- HPS Side
            ------------------------------------
            -- DDR3 SDRAM
            memory_mem_a        => HPS_DDR3_ADDR_o,
            memory_mem_ba       => HPS_DDR3_BA_o,
            memory_mem_ck       => HPS_DDR3_CK_P_o,
            memory_mem_ck_n     => HPS_DDR3_CK_N_o,
            memory_mem_cke      => HPS_DDR3_CKE_o,
            memory_mem_cs_n     => HPS_DDR3_CS_N_o,
            memory_mem_ras_n    => HPS_DDR3_RAS_N_o,
            memory_mem_cas_n    => HPS_DDR3_CAS_N_o,
            memory_mem_we_n     => HPS_DDR3_WE_N_o,
            memory_mem_reset_n  => HPS_DDR3_RESET_N_o,
            memory_mem_dq       => HPS_DDR3_DQ_io,
            memory_mem_dqs      => HPS_DDR3_DQS_P_io,
            memory_mem_dqs_n    => HPS_DDR3_DQS_N_io,
            memory_mem_odt      => HPS_DDR3_ODT_o,
            memory_mem_dm       => HPS_DDR3_DM_o,
            memory_oct_rzqin    => HPS_DDR3_RZQ_i,

            conduit_export_switch_i (9 downto 0)                => SW_i ,     -- switch_i
            conduit_export_switch_i (31 downto 10)                  => (others => '0'),
            conduit_export_key_i  (3 downto 0)                      => KEY_i,     -- key_i
            conduit_export_key_i (31 downto 4)                      => (others => '0'),

            conduit_export_leds_o (9 downto 0)                      => LEDR_o,     -- leds_o

            conduit_export_hex0_o (6 downto 0)                      => HEX0_o,     -- hex0_o
            conduit_export_hex1_o (6 downto 0)                      => HEX1_o,     -- hex1_o
            conduit_export_hex2_o (6 downto 0)                      => HEX2_o,     -- hex2_o
            conduit_export_hex3_o (6 downto 0)                      => HEX3_o,     -- hex3_o
            conduit_export_hex4_o (6 downto 0)                      => HEX4_o,     -- hex4_o
            conduit_export_hex5_o (6 downto 0)                  => HEX5_o,     -- hex5_o


            -- Pushbutton
            hps_io_hps_io_gpio_inst_GPIO54  => HPS_KEY_io,

            -- LED
            hps_io_hps_io_gpio_inst_GPIO53  => HPS_LED_io
    );

end top;
```

```c
/*******************************************************************************
 * HEIG-VD
 * Haute Ecole d'Ingenerie et de Gestion du Canton de Vaud
 * School of Business and Engineering in Canton de Vaud
 *******************************************************************************
 * REDS Institute
 * Reconfigurable Embedded Digital Systems
 *******************************************************************************
 *
 * File                   : labo5.c
 * Author                 : Spinelli Isaia
 * Date                   : 01.05.2020
 *
 * Context                : SOCF tutorial lab
 *
 *******************************************************************************
 * Brief: Programme for labo 5 of SOCF, for DE1-SoC board
 *
 *
 *******************************************************************************
 * Modifications :
 * Ver    Date        Student        Comments
 * 0.1    01.05.20    Isaia Spinelli : Modif pour la partie 1
 * 1.1    03.05.20    Isaia Spinelli : Ajout de la partie 2
*******************************************************************************/

#include "defines.h"


/* Variable globales */

int irqKey2 = 0;
int irqKey3 = 0;




int main(void){

    // tableau de converssion         0     1     2     3    4      5    6     7     8     9    a      b    c      d      e    f
    char tab_dec_to_hex_7seg[16] = {0x40, 0xF9, 0x24, 0x30, 0x19, 0x12, 0x02, 0xF8, 0x00, 0x10, 0x08, 0x03, 0x27, 0x21, 0x06, 0x0e };
    int led_tmp,Seg_tmp;


    /*---------- INTI ----------*/
    AXI_HEX5 = 0x40;
    AXI_HEX4 = 0xF9;
    AXI_HEX3 = 0x24;
    AXI_HEX2 = 0x30;
    AXI_HEX1 = 0x19;
    AXI_HEX0 = 0x02;

    AXI_LEDS = AXI_SWITCHES;

    unsigned int cst = AXI_REG_CONST;
    AXI_REG_TEST = cst;

    // Masque le bouton key3 (pour tester le masquage des interruptions)
    // AXI_INT_MASK = KEY3;

    disable_A9_interrupts();   // disable interrupts in the A9 processor
    set_A9_IRQ_stack();        // initialize the stack pointer for IRQ mode
    config_GIC();              // configure the general interrupt controller
    config_KEYs();             // configure KEYs to generate interrupts
    enable_A9_interrupts();    // enable interrupts in the A9 processor




    while(1){
        /* Appuie sur KEY 0*/
        if ((AXI_KEYS & KEY0) == 0) {
            // l'états des switches est copiés sur les LEDs.
            AXI_LEDS = AXI_SWITCHES;
            // Les afficheurs HEX5 à HEX0 affichent en hexadécimal les bits 23 à 0 de
            la constante définie dans l’IP.
            AXI_HEX0 = tab_dec_to_hex_7seg[cst & 0xF];
            AXI_HEX1 = tab_dec_to_hex_7seg[(cst>>4)  & 0xF];
            AXI_HEX2 = tab_dec_to_hex_7seg[(cst>>8)  & 0xF];
            AXI_HEX3 = tab_dec_to_hex_7seg[(cst>>12) & 0xF];
            AXI_HEX4 = tab_dec_to_hex_7seg[(cst>>16) & 0xF];
            AXI_HEX5 = tab_dec_to_hex_7seg[(cst>>20) & 0xF];


        /* Appuie sur KEY 1 */
        } else if ((AXI_KEYS & KEY1) == 0) {
            //  l'états inverses des switches est copiés sur les LEDs.
            AXI_LEDS = ~AXI_SWITCHES;

            // Les afficheurs HEX5 à HEX0 affichent en hexadécimal l’inverse des bits
            23 à 0 de la
            // constante définie dans l’IP.
            AXI_HEX0 = ~tab_dec_to_hex_7seg[cst & 0xF];
            AXI_HEX1 = ~tab_dec_to_hex_7seg[(cst>>4)  & 0xF];
            AXI_HEX2 = ~tab_dec_to_hex_7seg[(cst>>8)  & 0xF];
            AXI_HEX3 = ~tab_dec_to_hex_7seg[(cst>>12) & 0xF];
            AXI_HEX4 = ~tab_dec_to_hex_7seg[(cst>>16) & 0xF];
            AXI_HEX5 = ~tab_dec_to_hex_7seg[(cst>>20) & 0xF];

        // Si le bouton 2 est pressé (via une interruption)
        } else if (irqKey2) {
            irqKey2 = 0;

            /*  l’affichage des LEDs et des afficheurs 7 segments subit unerotation à
            droite */
            led_tmp = AXI_LEDS & 0x1;
            AXI_LEDS = ((AXI_LEDS & 0x3ff) >> 1) | (led_tmp << 9);

            Seg_tmp = AXI_HEX0;
            AXI_HEX0 = AXI_HEX1;
            AXI_HEX1 = AXI_HEX2;
            AXI_HEX2 = AXI_HEX3;
            AXI_HEX3 = AXI_HEX4;
            AXI_HEX4 = AXI_HEX5;
            AXI_HEX5 = Seg_tmp;


        // Si le bouton 3 est pressé (via une interruption)
        } else if (irqKey3) {
            irqKey3 = 0;

            /* l’affichage des LEDs et des afficheurs 7 segments subit une rotation à
            gauche */
            led_tmp = AXI_LEDS & 0x200;
            AXI_LEDS = (AXI_LEDS << 1) | (led_tmp >> 9);

            Seg_tmp = AXI_HEX5;
            AXI_HEX5 = AXI_HEX4;
            AXI_HEX4 = AXI_HEX3;
            AXI_HEX3 = AXI_HEX2;
            AXI_HEX2 = AXI_HEX1;
            AXI_HEX1 = AXI_HEX0;
            AXI_HEX0 = Seg_tmp;


        }

        AXI_HEX5 = test1;
    }
    AXI_HEX5 = test1;

}

/* Routine d'interruption */
void pushbutton_ISR(void){
    // Permet de tester le masquage
    // static int cpt_int = 0;

    /* Lecture et acquitement des interruptions */
    int src_irq = AXI_INT_SRC;

    // Key2 pressé
    if (src_irq & KEY2) {
        irqKey2 = 1;
    }

    // Key3 pressé
    if (src_irq & KEY3) {
        irqKey3 = 1;
    }


    // Tous les 3 interruptions de KEY0 et KEY1, change le masque de key 2 et 3
    /*
    if (src_irq & KEY0 || src_irq & KEY1) {
        cpt_int++;

        if (cpt_int % 3 == 0)
            AXI_INT_MASK = AXI_INT_MASK ^ (KEY3 | KEY2);
    }
    */

}

```

```c
/*******************************************************************************
 * HEIG-VD
 * Haute Ecole d'Ingenerie et de Gestion du Canton de Vaud
 * School of Business and Engineering in Canton de Vaud
 *******************************************************************************
 * REDS Institute
 * Reconfigurable Embedded Digital Systems
 *******************************************************************************
 *
 * File                 : defines.h
 * Author               : Sébastien Masle
 * Date                 : 16.02.2018
 *
 * Context              : SOCF class
 *
 *******************************************************************************
 * Brief: some definitions
 *
 *******************************************************************************
 * Modifications :
 * Ver    Date        Engineer      Comments
 * 0.0    16.02.2018  SMS           Initial version.
 * 1.1    06.05.20    Isaia Spinelli : Refactor
*******************************************************************************/

#include "exceptions.h"

// Déclaration de fonction
void pushbutton_ISR(void);

// Defines

#define     EDGE_TRIGGERED          0x1
#define     LEVEL_SENSITIVE         0x0
#define     CPU0                    0x01    // bit-mask; bit 0 represents cpu0
#define     ENABLE                  0x1

#define     USER_MODE               0b10000
#define     FIQ_MODE                0b10001
#define     IRQ_MODE                0b10010
#define     SVC_MODE                0b10011
#define     ABORT_MODE              0b10111
#define     UNDEF_MODE              0b11011
#define     SYS_MODE                0b11111

#define     INT_ENABLE              0b01000000
#define     INT_DISABLE             0b11000000

// Valeur des keys
#define KEY0 0x01
#define KEY1 0x02
#define KEY2 0x04
#define KEY3 0x08

// Typedef
typedef volatile unsigned char vcint;
typedef volatile unsigned short vsint;
typedef volatile unsigned int vuint;

```

```c
// Adresses
#define FPGA_BASE_ADDR_IO       0xFF200000
#define AXI_LIGHT_BASE_ADDR     FPGA_BASE_ADDR_IO


#define AXI_REG_CONST_CHAR      *(vcint *)(AXI_LIGHT_BASE_ADDR + 0x0)
#define AXI_REG_CONST_SHORT     *(vsint *)(AXI_LIGHT_BASE_ADDR + 0x0)
#define AXI_REG_CONST           *(vuint *)(AXI_LIGHT_BASE_ADDR + 0x0)

#define AXI_REG_TEST            *(vuint *)(AXI_LIGHT_BASE_ADDR + 0x4)

#define AXI_LEDS                *(vuint *)(AXI_LIGHT_BASE_ADDR + 0x100)

#define AXI_KEYS                *(vuint *)(AXI_LIGHT_BASE_ADDR + 0x200)
// Lecture de la source d'int. + acquitement
#define AXI_INT_SRC             *(vuint *)(AXI_LIGHT_BASE_ADDR + 0x204)
// 1 = interruption masquée
#define AXI_INT_MASK            *(vuint *)(AXI_LIGHT_BASE_ADDR + 0x208)

#define AXI_SWITCHES            *(vuint *)(AXI_LIGHT_BASE_ADDR + 0x300)

#define AXI_HEX0                *(vuint *)(AXI_LIGHT_BASE_ADDR + 0x400)
#define AXI_HEX1                *(vuint *)(AXI_LIGHT_BASE_ADDR + 0x410)
#define AXI_HEX2                *(vuint *)(AXI_LIGHT_BASE_ADDR + 0x420)
#define AXI_HEX3                *(vuint *)(AXI_LIGHT_BASE_ADDR + 0x430)
#define AXI_HEX4                *(vuint *)(AXI_LIGHT_BASE_ADDR + 0x440)
#define AXI_HEX5                *(vuint *)(AXI_LIGHT_BASE_ADDR + 0x450)

```

```c
/*****************************************************************************************
 * HEIG-VD
 * Haute Ecole d'Ingenerie et de Gestion du Canton de Vaud
 * School of Business and Engineering in Canton de Vaud
 *****************************************************************************************
 * REDS Institute
 * Reconfigurable Embedded Digital Systems
 *****************************************************************************************
 *
 * File                  : exceptions.h
 * Author                : Isaia Spinelli
 * Date                  : 06.05.2020
 *
 * Context               : SOCF class
 *
 *****************************************************************************************
 * Modifications :
 * Ver    Date        Engineer      Comments
 * 1.1    06.05.20    Isaia Spinelli : Refactor
 *
*****************************************************************************************/

void disable_A9_interrupts (void);
void set_A9_IRQ_stack (void);
void config_GIC (void);
void config_KEYs (void);
void enable_A9_interrupts (void);
void config_interrupt (int, int);
```

```c
/*******************************************************************************
 * HEIG-VD
 * Haute Ecole d'Ingenerie et de Gestion du Canton de Vaud
 * School of Business and Engineering in Canton de Vaud
 *******************************************************************************
 * REDS Institute
 * Reconfigurable Embedded Digital Systems
 *******************************************************************************
 *
 * File                 : execptions.c
 * Author               : Sébastien Masle
 * Date                 : 16.02.2018
 *
 * Context              : SOCF class
 *
 *******************************************************************************
 * Brief: defines exception vectors for the A9 processor
 *        provides code that sets the IRQ mode stack, and that dis/enables interrupts
 *        provides code that initializes the generic interrupt controller
 *
 *******************************************************************************
 * Modifications :
 * Ver    Date        Engineer      Comments
 * 0.0    16.02.2018  SMS           Initial version.
 * 1.0    13.03.2020  Spinelli Isaia
 *
*******************************************************************************/
#include <stdint.h>

#include "address_map_arm.h"
#include "defines.h"




// Référence : Exemple dans Using The ARM Generic

// Define the IRQ exception handler
void __attribute__ ((interrupt)) __cs3_isr_irq(void)
{
    /***********
     * Attention dans Qsys mettre sur flanc et non level !
     **********/

    // Read CPU Interface registers to determine which peripheral has caused an interrupt
    int interrupt_ID =*((int*) 0xFFFEC10C);

    // Handle the interrupt if it comes from the KEYs
    if (interrupt_ID == 72) {
        pushbutton_ISR();
    } else {
        while (1);                     // if unexpected, then stay here
    }

    // Clear interrupt from the CPU Interface
    *((int*) 0xFFFEC110) = interrupt_ID;

    return;
```

```c
}

// Define the remaining exception handlers
void __attribute__ ((interrupt)) __cs3_reset (void)
{
    while(1);
}

void __attribute__ ((interrupt)) __cs3_isr_undef (void)
{
    while(1);
}

void __attribute__ ((interrupt)) __cs3_isr_swi (void)
{
    while(1);
}

void __attribute__ ((interrupt)) __cs3_isr_pabort (void)
{
    while(1);
}

void __attribute__ ((interrupt)) __cs3_isr_dabort (void)
{
    while(1);
}

void __attribute__ ((interrupt)) __cs3_isr_fiq (void)
{
    while(1);
}

/*
 * Initialize the banked stack pointer register for IRQ mode
*/
void set_A9_IRQ_stack(void)
{
    uint32_t stack, mode;
    stack = A9_ONCHIP_END - 7;      // top of A9 onchip memory, aligned to 8 bytes
    /* change processor to IRQ mode with interrupts disabled */
    mode = INT_DISABLE | IRQ_MODE;
    asm("msr cpsr, %[ps]" : : [ps] "r" (mode));
    /* set banked stack pointer */
    asm("mov sp, %[ps]" : : [ps] "r" (stack));

    /* go back to SVC mode before executing subroutine return! */
    mode = INT_DISABLE | SVC_MODE;
    asm("msr cpsr, %[ps]" : : [ps] "r" (mode));
}

/*
 * Turn on interrupts in the ARM processor
*/
void enable_A9_interrupts(void)
{
    uint32_t status = SVC_MODE | INT_ENABLE;
    asm("msr cpsr, %[ps]" : : [ps]"r"(status));
}

/** Turn off interrupts in the ARM processor*/
void disable_A9_interrupts(void) {
    int status = 0b11010011;
    asm("msr cpsr, %[ps]" : : [ps]"r"(status));
}

void config_GIC (void) {
     // configure the FPGA KEYs interrupt (72)
    config_interrupt (72, 1);

    // Set Interrupt Priority Mask Register (ICCPMR). Enable all priorities
    *((int*) 0xFFFEC104) = 0xFFFF;

    // Set the enable in the CPU Interface Control Register (ICCICR)
    *((int*) 0xFFFEC100) = 1;

    // Set the enable in the Distributor Control Register (ICDDCR)
    *((int*) 0xFFFED000) = 1;
}

void config_KEYs (void) {
    volatile int*KEY_ptr = (int*) 0xFF200050;   // KEY base address

    *(KEY_ptr + 2) = 0xF;    // enable interrupts for all four KEYs

}

void config_interrupt (int N, int CPU_target) {
    int reg_offset, index, value, address;

    /*Configure the Interrupt Set-Enable Registers (ICDISERn).
     *reg_offset = (integer_div(N / 32)*4; value = 1 << (N mod 32)*/

    reg_offset = (N >> 3) & 0xFFFFFFFC;
    index = N & 0x1F;
    value = 0x1 << index;
    address = 0xFFFED100 + reg_offset;

    /*Using the address and value, set the appropriate bit*/
    *(int*)address |= value;

    /*Configure the Interrupt Processor Targets Register (ICDIPTRn)
     * reg_offset = integer_div(N / 4)*4; index = N mod 4*/
    reg_offset = (N & 0xFFFFFFFC);
    index = N & 0x3;
    address = 0xFFFED800 + reg_offset + index;

    /*Using the address and value, write to (only) the appropriate byte*/
    *(char*)address = (char) CPU_target;
}

```

```c
/*****************************************************************************************
 * HEIG-VD
 * Haute Ecole d'Ingenerie et de Gestion du Canton de Vaud
 * School of Business and Engineering in Canton de Vaud
 *****************************************************************************************
 * REDS Institute
 * Reconfigurable Embedded Digital Systems
 *****************************************************************************************
 *
 * File                   : address_map_arm.h
 * Author                 : Sébastien Masle
 * Date                   : 16.02.2018
 *
 * Context                : SOCF class
 *
 *****************************************************************************************
 * Brief: provides address values that exist in the system
 *
 *****************************************************************************************
 * Modifications :
 * Ver    Date        Engineer      Comments
 * 0.0    16.02.2018  SMS           Initial version.
 *
******************************************************************************************/

#define BOARD                                "DE1-SoC"

/* Memory */
#define DDR_BASE                             0x00000000
#define DDR_END                              0x3FFFFFFF
#define A9_ONCHIP_BASE                       0xFFFF0000
#define A9_ONCHIP_END                        0xFFFFFFFF
#define SDRAM_BASE                           0xC0000000
#define SDRAM_END                            0xC3FFFFFF
#define FPGA_ONCHIP_BASE                     0xC8000000
#define FPGA_ONCHIP_END                      0xC803FFFF
#define FPGA_CHAR_BASE                       0xC9000000
#define FPGA_CHAR_END                        0xC9001FFF

```