Annexe

```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  08.05.2029  PMR    Complétion du laboratoire
------------------------------------------------------------------------------


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 (
        axi_clk_i       : in  std_logic;
        axi_reset_i     : in  std_logic;
        -- AXI4-Lite
        axi_awaddr_i    : in  std_logic_vector(AXI_ADDR_WIDTH-1 downto 0);
        axi_awprot_i    : in  std_logic_vector( 2 downto 0); -- Pas utilisés
        axi_awvalid_i   : in  std_logic;
        axi_awready_o   : out std_logic;
        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;
        axi_bresp_o     : out std_logic_vector(1 downto 0);
        axi_bvalid_o    : out std_logic;
        axi_bready_i    : in  std_logic;
        axi_araddr_i    : in  std_logic_vector(AXI_ADDR_WIDTH-1 downto 0);
        axi_arprot_i    : in  std_logic_vector( 2 downto 0); -- Pas utilisés
        axi_arvalid_i   : in  std_logic;
        axi_arready_o   : out std_logic;
        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

        --TEST POUR TESTBENCH
--          vect_input_A_i  : in std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
--          vect_input_B_i  : in std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
--          vect_input_C_i  : in std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
--          vect_input_D_i  : in std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
--
--          output_reg_A_o  : out std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
--          output_reg_B_o  : out std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
--          output_reg_C_o  : out std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
--          output_reg_D_o  : out std_logic_vector(AXI_DATA_WIDTH-1 downto 0)


        switch_i      : in std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
        keys_i      : in std_logic_vector(AXI_DATA_WIDTH-1 downto 0);

        leds_o      : out std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
        hex03_o      : out std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
        hex54_o      : out std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
        -- Interruptions

        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;

    --signal for the AXI slave
    --intern signal for output
    signal axi_awready_s       : std_logic;
    signal axi_wready_s        : std_logic;
    signal axi_arready_s       : std_logic;
    signal axi_rvalid_s        : std_logic;
    signal axi_bvalid_s        : std_logic;
    signal axi_bresp_s         : std_logic_vector(1 downto 0);
    signal axi_rdata_s          : std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
    signal axi_wdata_s    : std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
     --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_data_wren_s     : std_logic;
    signal axi_data_reen_s     : std_logic;
```

```
     -- signal representing registers
    signal const_register_s      : std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
    signal test_register_s      : std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
    signal leds_register_s      : std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
    signal hex03_register_s      : std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
    signal hex54_register_s      : std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
    signal switch_register_s      : std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
    signal keys_register_s      : std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
    signal keys_im_register_s   : std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
    signal keys_ec_register_s   : std_logic_vector(AXI_DATA_WIDTH-1 downto 0);



    -- signaux permettant la gestion du strobe
    signal temp_Vect_Strb_1_s : std_logic_vector(AXI_DATA_WIDTH/4 -1 downto 0);
    signal temp_Vect_Strb_2_s : std_logic_vector(AXI_DATA_WIDTH/4 -1 downto 0);
    signal temp_Vect_Strb_3_s : std_logic_vector(AXI_DATA_WIDTH/4 -1 downto 0);
    signal temp_Vect_Strb_4_s : std_logic_vector(AXI_DATA_WIDTH/4 -1 downto 0);

    -- signal d'interruption
    signal irq_s   : std_logic;

    -- signaux de masquage pour la lecture des registres
    signal KEY_MASK_s    : std_logic_vector(AXI_DATA_WIDTH - 1 downto 0);
    signal SWITCH_LEDS_MASK_s : std_logic_vector(AXI_DATA_WIDTH - 1 downto 0);
    signal HEX54_MASK_s   : std_logic_vector(AXI_DATA_WIDTH - 1 downto 0);
    signal HEX03_MASK_s   : std_logic_vector(AXI_DATA_WIDTH - 1 downto 0);

--intern signal for adress decoding (Prévus dans la première version)

--signal local_address_write_s      : integer;
--signal local_address_read_s       : integer;
--signal const_register_address_valid_write_s : std_logic;
--signal test_register_address_valid_write_s : std_logic;
--signal leds_register_address_valid_write_s : std_logic;
--signal hex03_register_address_valid_write_s : std_logic;
--signal hex45_register_address_valid_write_s : std_logic;
--signal switch_register_address_valid_write_s : std_logic;
--signal keys_register_address_valid_write_s : std_logic;
--signal const_register_address_valid_read_s : std_logic;
--signal test_register_address_valid_read_s : std_logic;
--signal leds_register_address_valid_read_s : std_logic;
--signal hex03_register_address_valid_read_s : std_logic;
--signal hex45_register_address_valid_read_s : std_logic;
--signal switch_register_address_valid_read_s : std_logic;
--signal keys_register_address_valid_read_s : std_logic;




begin

    reset_s  <= axi_reset_i;




-- Ce code ci-dessous présente le premier essai basé sur le document présentant
l'implémentation d'une IP axi_lite
```

```
-----------------------------------------------------------
-- address decoding

-- On récupére l'offset dans l'adresse. On utilise pas les deux bits de poid
faible
--local_address_write_s <= to_integer(unsigned(axi_waddr_mem_s(AXI_ADDR_WIDTH -
1 downto 0)));
--local_address_read_s <= to_integer(unsigned(axi_araddr_mem_s(AXI_ADDR_WIDTH -
1 downto 0)));

--address_range_analysis : process (local_address_write_s, local_address_read_s)
--begin
--  -- write valid
--  const_register_address_valid_write_s <= '0';
--  test_register_address_valid_write_s <= '0';
--  leds_register_address_valid_write_s <= '0';
--  hex03_register_address_valid_write_s <= '0';
--  hex45_register_address_valid_write_s <= '0';
--  switch_register_address_valid_write_s <= '0';
--  keys_register_address_valid_write_s <= '0';
--
--  -- read valid
--  const_register_address_valid_read_s <= '0';
--  test_register_address_valid_read_s <= '0';
--  leds_register_address_valid_read_s <= '0';
--  hex03_register_address_valid_read_s <= '0';
--  hex45_register_address_valid_read_s <= '0';
--  switch_register_address_valid_read_s <= '0';
--  keys_register_address_valid_read_s <= '0';
--
--  case (local_address_write_s) is
--      when 0 => const_register_address_valid_write_s <= '1';
--      when 1 => test_register_address_valid_write_s <= '1';
--      when 2 => leds_register_address_valid_write_s <= '1';
--      when 3 => hex03_register_address_valid_write_s <= '1';
--      when 4 => hex45_register_address_valid_write_s <= '1';
--      when 5 => switch_register_address_valid_write_s <= '1';
--      when 6 => keys_register_address_valid_write_s <= '1';
--      when others => NULL;
--  end case;
--
--  case (local_address_read_s) is
--      when 0 => const_register_address_valid_read_s <= '1';
--      when 1 => test_register_address_valid_read_s <= '1';
--      when 2 => leds_register_address_valid_read_s <= '1';
--      when 3 => hex03_register_address_valid_read_s <= '1';
--      when 4 => hex45_register_address_valid_read_s <= '1';
--      when 5 => switch_register_address_valid_read_s <= '1';
--      when 6 => keys_register_address_valid_read_s <= '1';
--      when others => NULL;
--  end case;
--end process;
--
-----------------------------------------------------------
-- Write address channel

    -- GESTION DU STROBE
```

```vhdl
    -- On récupère les bits voulus en fonction du strobe et on assemble les
résultats dans un signal temporaire (axi_wdata_s)
    temp_Vect_Strb_1_s <= axi_wdata_i(7 downto 0) when axi_wstrb_i(0) = '1' else
(others => '0');
    temp_Vect_Strb_2_s <= axi_wdata_i(15 downto 8) when axi_wstrb_i(1) = '1'
else (others => '0');
    temp_Vect_Strb_3_s <= axi_wdata_i(23 downto 16) when axi_wstrb_i(2) = '1'
else (others => '0');
    temp_Vect_Strb_4_s <= axi_wdata_i(31 downto 24) when axi_wstrb_i(3) = '1'
else (others => '0');
    axi_wdata_s <= temp_Vect_Strb_4_s & temp_Vect_Strb_3_s & temp_Vect_Strb_2_s
& temp_Vect_Strb_1_s;


    process (reset_s, axi_clk_i)
    begin
        if reset_s = '1' then
            axi_awready_s <= '0';
            axi_waddr_mem_s <= (others => '0');
        elsif rising_edge(axi_clk_i) then
            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';
            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
        if reset_s = '1' then
            axi_wready_s    <= '0';
        elsif rising_edge(axi_clk_i) then
            if(axi_wready_s = '0' and axi_wvalid_i = '1') then
                axi_wready_s <= '1';
            else
                axi_wready_s <= '0';
            end if;
        end if;
    end process;

    -- Assignation de la sortie axi_wready_o
    axi_wready_o <= axi_wready_s;
    --condition to write data. On peut écrire des données quand axi_wready_s et
axi_wvalid_i sont actif (Selon documentation axi_lite)
    axi_data_wren_s <= '0' when (axi_wready_s = '1' and axi_wvalid_i = '1') else
                       '1';


    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
            --Assignation des valeurs de base aux registres et au signal
d'interruption lors du reset
            test_register_s <= (others => '0');--vect_input_A_i;
            const_register_s <= x"DEADBEEF";
            leds_register_s <= x"00000000";
            hex03_register_s <= x"00000000";
            hex54_register_s <= x"00000000";
            switch_register_s <= switch_i;--vect_input_B_i;
            keys_register_s <=  keys_i;--vect_input_C_i;
            keys_im_register_s <= (others => '0');
            keys_ec_register_s <= (others => '0');
            irq_s <= '0';

             -- création des masques
            KEY_MASK_s <= (others => '0');
            SWITCH_LEDS_MASK_s <= (others => '0');
            HEX03_MASK_s <= (others => '1');
            HEX54_MASK_s <= (others => '1');

            KEY_MASK_s(3 downto 0) <= (others => '1');
            SWITCH_LEDS_MASK_s(9 downto 0) <= (others => '1');
            HEX03_MASK_s(31) <= '0';
            HEX03_MASK_s(23) <= '0';
            HEX03_MASK_s(15) <= '0';
            HEX03_MASK_s(7)  <= '0';
            HEX54_MASK_s(31 downto 15) <= (others => '0');
            HEX54_MASK_s(7)  <= '0';
        elsif rising_edge(axi_clk_i) then
            -- Récupération de la valeur des switchs dans le registre du bus
axi_lite correspondant
            switch_register_s <= switch_i;

            -- GESTION DE L'INTERRUPTION
            -- Si un bouton à été appuyé, on regarde quel bouton est actif, on
s'assure que le registre de masque d'interruption est
            -- actif pour le bit correspondant au bouton actif, on remplit le
bit du registre d'edge capture correspondant et on active
            -- le signal générant l'irq
            if(keys_register_s /= keys_i) then
                if(keys_register_s(0) = '0' and keys_i(0) = '1' and
(keys_im_register_s(0) = '1')) then
                    keys_ec_register_s(0) <= '1';
                    irq_s <= '1';
                end if;
                if(keys_register_s(1) = '0' and keys_i(1) = '1' and
(keys_im_register_s(1) = '1')) then
                    keys_ec_register_s(1) <= '1';
                    irq_s <= '1';
                end if;
                if(keys_register_s(2) = '0' and keys_i(2) = '1' and
(keys_im_register_s(2) = '1')) then
                    keys_ec_register_s(2) <= '1';
                    irq_s <= '1';
                end if;
                if(keys_register_s(3) = '0' and keys_i(3) = '1' and
(keys_im_register_s(3) = '1')) then
                    keys_ec_register_s(3) <= '1';
                    irq_s <= '1';
                end if;
            end if;

            -- Récupération de la valeur des boutons dans le registre du bus
axi_lite correspondant
            keys_register_s <= keys_i;

            -- Si on peut écrire
            if axi_data_wren_s = '0' then

                -- On transforme l'adresse mémorisée en entier et on effectue
l'écriture souhaitée en fonction de l'addresse.
                -- J'ai choisit d'appliquer ici un masque assurant que la valeur
écrite suive le plan d'addressage et permettant
                -- de contenir dans les registres les valeurs souhaitées lors
par la suite de la lecture du registre selon le plan
                -- d'adressage. Ce traitement pourrait être fait dans la partie
lecture, mais je préfère avoir le contrôle sur ce
                -- qui est écrit dans mes registres plutôt que de renvoyer lors
de la lecture un contenu modifié du registre afin
                -- de coller au plan d'addressage (Lecture de '0' pour certains
bits). De plus, la gestion des lectures avec les
                -- bits à '0' pour les keys et les switchs est gérée directement
dans le top de la DE1-SOC
                int_waddr_v   := to_integer(unsigned(axi_waddr_mem_s));
                case int_waddr_v is
                    when 0   => null; -- constante, on écrit pas dedans
                    when 1   => test_register_s <= axi_wdata_s; -- Test register
                    when 2   => leds_register_s <= axi_wdata_s and
SWITCH_LEDS_MASK_s; -- Leds register
                    when 3   => hex03_register_s <= axi_wdata_s and
HEX03_MASK_s; -- HEX3..0 register
                    when 4   => hex54_register_s <= axi_wdata_s and
HEX54_MASK_s; -- HEX5..4 register
                    when 7   => keys_im_register_s <= axi_wdata_s and
KEY_MASK_s;
                    when 8   => keys_ec_register_s <= (others => '0');irq_s <=
'0';
                    when others => null;  --on écrit pas dedans
                end case;
            end if;
        end if;
    end process;

    -- Assignation de l'interruption
    irq_o <= irq_s;
-----------------------------------------------------------
```

```
-- Write response channel

   process (reset_s , axi_clk_i)
   begin
    if reset_s = '1' then
        -- Assigniation des valeurs de base pour les signaux du canal réponse de
write
        axi_bvalid_s <= '0';
        axi_bresp_s <= "00";
    elsif rising_edge(axi_clk_i) then
        -- Si on peut écrire, on met le signal bvalid_s actif. La réponse sera
toujours 00 dans notre cas. Je laisse l'assigniation de
        -- la réponse ici dans le cas d'une future modification.
        if axi_data_wren_s = '0' then
            axi_bvalid_s <= '1';
            axi_bresp_s <= "00";
        else
            axi_bvalid_s <= '0';
        end if;
    end if;
   end process;

   -- Assignation des signaux de réponses
   axi_bvalid_o <= axi_bvalid_s;
   axi_bresp_o <= axi_bresp_s;



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

    process (reset_s, axi_clk_i)
    begin
        if reset_s = '1' then
           axi_arready_s    <= '0';
           axi_araddr_mem_s <= (others => '1');
        elsif rising_edge(axi_clk_i) then
            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;

    axi_arready_o <= axi_arready_s;

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


    process (reset_s, axi_clk_i)
        --number address to access 32 or 64 bits data
        variable int_raddr_v : natural;
    begin
        if reset_s = '1' then

          axi_rdata_s <= (others => '0');
          axi_rvalid_s    <= '0';
        elsif rising_edge(axi_clk_i) then
            -- Afin de respecter les timings, pour pouvoir remplir le signal
axi_rdata_s, on doit s'assurer que axi_arready_s et axi_arvalid_i soit à 1
            -- si c'est le cas, on met axi_rvalid_s actif, on récupère l'adresse
recu pour la lecture et on remplit le axi_rdata_s avec le
            -- contenu du registre correspondant.
            if axi_arready_s = '1' and axi_arvalid_i = '1' then
                axi_rvalid_s <= '1';
                int_raddr_v   := to_integer(unsigned(axi_araddr_mem_s));
                case int_raddr_v is
                    when 0   => axi_rdata_s <= const_register_s; -- constante,
on écrit pas dedans
                    when 1   => axi_rdata_s <= test_register_s; -- Test register
                    when 2   => axi_rdata_s <= leds_register_s; -- Leds register
                    when 3   => axi_rdata_s <= hex03_register_s; -- HEX3..0
register
                    when 4   => axi_rdata_s <= hex54_register_s; -- HEX5..4
register
                    when 5   => axi_rdata_s <= switch_register_s; -- switch
register
                    when 6   => axi_rdata_s <= keys_register_s;   -- key
register
                    when 7   => axi_rdata_s <= keys_im_register_s; -- interrupt
mask register
                    when 8   => axi_rdata_s <= keys_ec_register_s; -- edge
capture register
                    when others => null;    -- Dans les autres cas, on ne fait
rien
                end case;
            else
                axi_rvalid_s <= '0';
            end if;
        end if;
    end process;

   -- Assignation des sorties des canaux de lectures
   axi_rvalid_o <= axi_rvalid_s;
   axi_rdata_o <= axi_rdata_s;
   axi_rresp_o <= "00";


   -- Assignation pour test-bench
--     output_reg_A_o <= const_register_s; --const_register_s;
--     output_reg_B_o <= test_register_s;
--     output_reg_C_o <= leds_register_s;
--     output_reg_D_o <= hex03_register_s;

    -- Assignation des I/O
    leds_o      <= leds_register_s;
    hex03_o     <= hex03_register_s;
    hex54_o     <= hex54_register_s;
```

```
end rtl;
```