Final Project/Exam
EE365
Custom IP Module Integration

Ernesto Sola-Thomas
December 13th, 2022

This project tasked students to explore and implement the diverse tools that Vivado has to offer while making custom IPs. There where a total of 4 IPs made for this project. Two of them were replicated from some online tutorials and the other two IPs were created using VHDL designs that were developed through the course of this class and these designs have been thoroughly tested. This IPs include an LCD interface using an 8 bit data bus and a 7-Segment display interface implementing the Serial TTL protocol.

The first followed tutorial was provided by Digilent and it covered how to create a custom PWM interface to control some LEDs with a PWM signal. This tutorial was particularly useful for the rest of the project because it covered and showed step by step how to connect the FPGA to the Hard processor of the chip by creating a bridge using an AXI interface, this tutorial showed how and where to write in the memory of the device to be able to send data from the Hard processor executing software instructions to the implemented hardware in the FPGA. The practice I obtained from working through the steps of this tutorial was of great importance through the rest parts of this project.

The second followed tutorial was useful to test the knowledge gathered from the first tutorial and another big thing that I was able to get out of it was how to use both interfaces the UART to communicate to the computer and the AXI bridge to communicate between the hard processor executing software instructions. This tutorial also allowed me to dive deeper into the file structure of custom IPs in Vivado as everything is stored in the ip_repo file that is created one directory above the current working directory. I realized this as I developed both tutorials in different directories and then I was not able to see the ip that I generated for the last ip. This could be fixed by assigning a master ip_repo directory path in my device and store in there all the ips that I generate.

After having finished both tutorials my lab partner and I proceeded to start developing our own custom-made IPs from the hardware designs we developed throughout the semester, figure 1 shows the block diagram design created for the 7-Segment display ip. The block has inputs for a asynchronous reset, a clock and the input for the bridge interface to receive data for the hard processor, that is the S00_AXI port. As show in the tutorials after creating the ip in Vivado and importing the VHDL designs that were previously created the wrapper VHDL files from the custom ip were modified to add the required output port and make the required connections between the different files in the hierarchy. Figure 2 shows the VHDL hierarchy of the 7-segment display IP, the first file “Serial_7seg_v1_0” is the top level of the hierarchy this acts a wrapper for the “Serial_7seg_v1_0_S00_AXI_inst” file which is the VHDL file that creates all the required instances and connections. This file instances the previously created hardware files for 7-segment display interface.

Figure 1: 7-Segment display Block Diagram

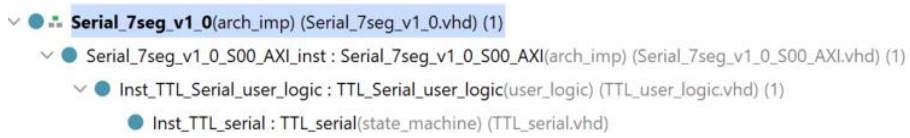

Figure 2: Hierarchy of 7-Segment display

The second generated IP is for the LCD interface, after having the practice of making three custom IPs developing this one was pretty straight forward. Figure 3 shows the block diagram of the LCD interface block, this block contains more inputs and outputs. This design gets its data through the S00_AXI bus as shown in the last custom design. This is a 32 bits bus by default and for the applications of this project only 16 bits are used so even though the S00_AXI bus carries 32 bits only 16 of them are accessed by the hardware. The outputs are directly tight to the hardware that was previously designed and tested. The only modification that had to be implemented was the instance of some LUTs that would transform the HEX values that are sent from the software to the required ASCII values that need to be sent to the LCD. Figure 4 shows the Hierarchy of the HDL files in this instance. Having the two wrapper layers that are created when working with Vivados custom IPs and under those are the custom VHDL files for LCD interface, the 4 instances for the HEX2ASCII module are used to convert the HEX to ASCII. There are 4 instances because each ASCII value is conformed by 4 bits and there are 16 bits transmitted from software to the FPGA so there is one instance of the HEX2ASCII per ASCII character sent and there is a multiplexer that selects which value to send to the LCD. The signal for this multiplexer changes at around 5 ms and sends the next value. The sent values also contain the required configuration commands for the LCD. The code for this section can be found in the Appendix of this document.

Figure 3: LCD interface Block Diagram

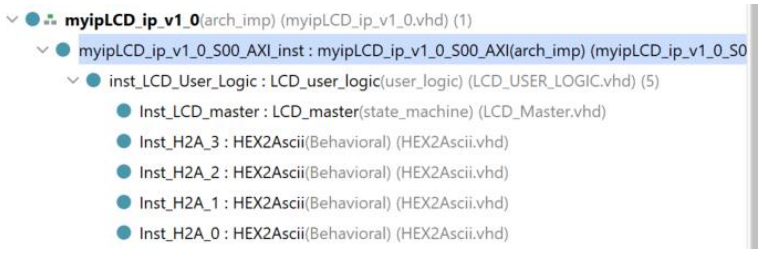

Figure 4: Hierarchy of LCD interface

After having successfully created all the IPs the interface between the hard processor and FPGA was created as shown in figure 5 the “px7_axi_peripherial” was created using the auto route tools that Viviado offers, this automatically assigned an address in memory for each AXI bridge. After having this module two prebuilt IPs where instantiated. One for a timer in software and another for an interrupt, this will be implemented in software to decide when the value that is sent to the custom IPs changes.

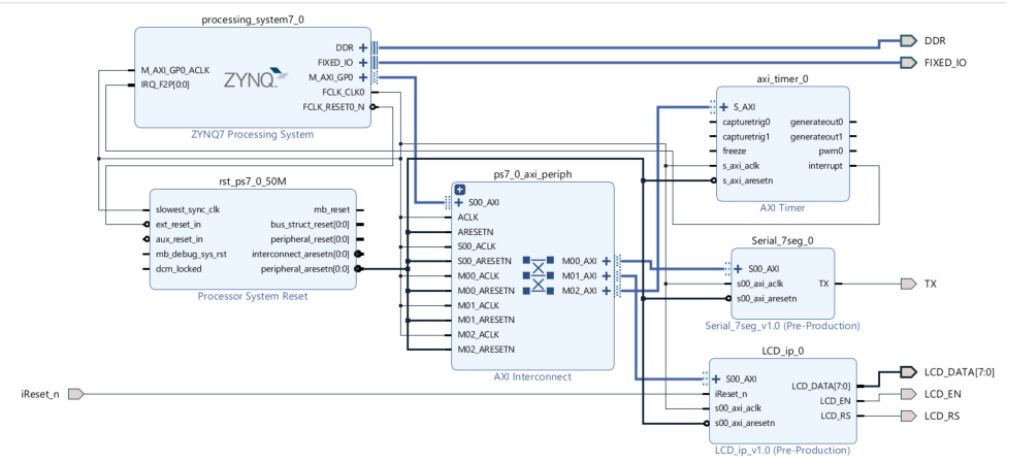

Figure 5: Block diagram of implemented design with both custom IPs

After having the hardware implementation, the bitsream was created and the hardware was exported to be able to work with the SDK. After executing the SDK and importing the timer example some changes where made to the example C file, these modifications can be found in the Appendix of this document. The address to which the Xil_Out32() function needs to write can be fount in the Adress manager tab of Vivado. The timer counter was set to trigger an interrupt once per second to then trigger a change in the written value of the memory address. The values written on the memory address cycle through 10 different hex values: 0x0000,0xFADE,0xCAFE,0x4B1D,0xFEED,0x1BAD,0xD00D,0xDEAD,0xBEEF,0xF00D

After having finished the implementations in the SDK the C code was compiled, the board was programed with the bitstream file, and the software application was executed. This proved that the custom IPs and their implementation was done successfully as the board is showing the expected values in the three peripherals. The UART, LCD and 7-segment display all of this peripherals iterate through the expected values at a rate of one second.

APENDIX:

LCD_USER_LOGIC

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
use IEEE.numeric_std.all;
--USE ieee.std_logic_unsigned.all;

ENTITY LCD_user_logic IS
  GENERIC (
    CONSTANT CntMax : integer := 49999);
  PORT(
```

```
    clk         : IN     STD_LOGIC;
    iReset_n    : IN     STD_LOGIC;
    SDK_message : IN     STD_LOGIC_VECTOR(15 DOWNTO 0);
    LCD_DATA  : OUT     STD_LOGIC_VECTOR(7 DOWNTO 0);
    LCD_EN  : OUT     STD_LOGIC;
    LCD_RS  : OUT    STD_LOGIC);
END LCD_user_logic;

ARCHITECTURE user_logic OF LCD_user_logic IS

TYPE state_type IS(start, ready, data_valid, busy_high, repeat); --needed states
signal state      : state_type;                    --state machine
signal pwrReset    : STD_LOGIC;                     -- reset delay
signal ena        : STD_LOGIC;                     --latch in data
signal temp_LCD_DATA       : STD_LOGIC_VECTOR(7 DOWNTO 0); --data to write
signal LCD_DATA_wr    : STD_LOGIC_VECTOR(7 DOWNTO 0); --data to write
signal temp_LCD_RS       : STD_LOGIC; --data to write
signal LCD_RS_wr    : STD_LOGIC; --data to write
signal busy       : STD_LOGIC;                     --indicates transaction in
signal count    : unsigned(27 DOWNTO 0):=X"000000F";
signal byteSel      : integer range 0 to 29:=0;
signal sig_LCD_DATA : std_logic_vector(7 DOWNTO 0);
signal sig_LCD_EN, sig_LCD_RS     : std_logic;
signal ASCII3, ASCII2, ASCII1, ASCII0  : std_logic_vector(7 DOWNTO 0);

COMPONENT LCD_master is
  Generic (Constant CntMax : integer:= 49999);  -- (125 MHz/500 KHz) - 1 = 249
  Port ( clock             : in std_logic; -- board clock
         iLCD_Data         : in std_logic_vector(7 downto 0);
         iLCD_RS           : in std_logic;
         resetDelay        : in std_logic;
         iReset_n          : in std_logic;
        iEna               : in std_logic;
        oBusy              : out std_logic;
        LCD_DATA           : out std_logic_vector(7 downto 0);
        LCD_EN, LCD_RS     : out std_logic
        );
end COMPONENT;

COMPONENT HEX2Ascii is
   port (
        HEX   : in std_logic_vector(3 downto 0);
        ASCII   : out std_logic_vector(7 downto 0)
    );
end Component;
```

```
BEGIN
LCD_DATA <=Sig_LCD_DATA;
LCD_RS <= Sig_LCD_RS;
LCD_EN <= Sig_LCD_EN;
process(byteSel)
 begin
    case byteSel is
       when 0  => temp_LCD_DATA <= X"38";
                  temp_LCD_RS <= '0';
       when 1  => temp_LCD_DATA <= X"38";
                  temp_LCD_RS <= '0';
       when 2  => temp_LCD_DATA <= X"38";
                  temp_LCD_RS <= '0';
       when 3  => temp_LCD_DATA <= X"38";
                  temp_LCD_RS <= '0';
       when 4  => temp_LCD_DATA <= X"38";
                  temp_LCD_RS <= '0';
       when 5  => temp_LCD_DATA <= X"38";
                  temp_LCD_RS <= '0';
       when 6  => temp_LCD_DATA <= X"01";
                  temp_LCD_RS <= '0';
       when 7  => temp_LCD_DATA <= X"0C";
                  temp_LCD_RS <= '0';
       when 8  => temp_LCD_DATA <= X"06";
                  temp_LCD_RS <= '0';
       when 9  => temp_LCD_DATA <= X"80";
                  temp_LCD_RS <= '0';
       when 10  => temp_LCD_DATA <= X"53";
                   temp_LCD_RS <= '1';
       when 11  => temp_LCD_DATA <= X"79";
                   temp_LCD_RS <= '1';
       when 12  => temp_LCD_DATA <= X"73";
                   temp_LCD_RS <= '1';
       when 13  => temp_LCD_DATA <= X"74";
                   temp_LCD_RS <= '1';
       when 14  => temp_LCD_DATA <= X"65";
                   temp_LCD_RS <= '1';
       when 15  => temp_LCD_DATA <= X"6D";
                   temp_LCD_RS <= '1';
       when 16  => temp_LCD_DATA <= X"FE";
                   temp_LCD_RS <= '1';
       when 17  => temp_LCD_DATA <= X"52";
                   temp_LCD_RS <= '1';
       when 18  => temp_LCD_DATA <= X"65";
```

```
                    temp_LCD_RS <= '1';
      when 19  => temp_LCD_DATA <= X"61";
                    temp_LCD_RS <= '1';
      when 20  => temp_LCD_DATA <= X"64";
                    temp_LCD_RS <= '1';
      when 21  => temp_LCD_DATA <= X"79";
                    temp_LCD_RS <= '1';
      when 22  => temp_LCD_DATA <= X"c0";    --repeat
                    temp_LCD_RS <= '0';
      when 23  => temp_LCD_DATA <= X"3D";
                    temp_LCD_RS <= '1';
      when 24  => temp_LCD_DATA <= X"3D";
                    temp_LCD_RS <= '1';
      when 25  => temp_LCD_DATA <= X"3E";
                    temp_LCD_RS <= '1';
      when 26  => temp_LCD_DATA <= ASCII3;   --ASCII3
                    temp_LCD_RS <= '1';
      when 27  => temp_LCD_DATA <= ASCII2;   --ASCII2
                    temp_LCD_RS <= '1';
      when 28  => temp_LCD_DATA <= ASCII1;   --ASCII1
                    temp_LCD_RS <= '1';
      when 29  => temp_LCD_DATA <= ASCII0;   --ASCII0
                    temp_LCD_RS <= '1';
      when others => temp_LCD_DATA <= X"38";
                     temp_LCD_RS <= '0';
   end case;
end process;

Inst_LCD_master: LCD_master
  GENERIC map(
    CntMax => 49999)
  port map(
    iReset_n  => iReset_n,
    resetDelay  => pwrReset,
    clock => clk,
    iEna  => ena,
    iLCD_RS => LCD_RS_wr,
    iLCD_DATA => LCD_DATA_wr,
        oBusy   => busy,
        LCD_DATA => Sig_LCD_DATA,
        LCD_RS => Sig_LCD_RS,
        LCD_EN => Sig_LCD_EN
    );

 Inst_H2A_3 : HEX2Ascii
```

```
    port map (
       HEX => SDK_Message(15 downto 12),
       ASCII => ASCII3
    );

  Inst_H2A_2 : HEX2Ascii
    port map (
       HEX => SDK_Message(11 downto 8),
       ASCII => ASCII2
    );

  Inst_H2A_1 : HEX2Ascii
    port map (
       HEX => SDK_Message(7 downto 4),
       ASCII => ASCII1
    );

 Inst_H2A_0 : HEX2Ascii
    port map (
       HEX => SDK_Message(3 downto 0),
       ASCII => ASCII0
    );

process(clk, iReset_n)
begin
if(clk'event and clk = '1') then
  case state is
  when start =>
      if count /= X"0000000" then
    count   <= count - 1;
    pwrReset <= '0';
    state   <= start;
    ena   <= '0';
  else
    pwrReset <= '1';
    state   <= ready;
    LCD_RS_wr <= temp_LCD_RS;
    LCD_DATA_wr <= temp_LCD_DATA;                  --data to be written
    end if;

  when ready =>
        if busy = '0' then
          ena     <= '1';
          state   <= data_valid;
        end if;
```

```
  when data_valid =>                                --state for conducting this
transaction
              if busy = '1' then
          ena     <= '0';
          state   <= busy_high;
              end if;
  when busy_high =>
              if iReset_n = '0' then
              byteSel <= 0;
              end if;
              if(busy = '0') then                -- busy just went low
          state <= repeat;
          end if;

  when repeat =>
            if byteSel < 29 then
               byteSel <= byteSel + 1;
          else
               byteSel <= 22;
          end if;
              state <= start;
  when others => null;

  end case;
end if;
end process;
end user_logic;
```

HEX2Ascii

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

entity HEX2Ascii is
    port (
        HEX  : in std_logic_vector(3 downto 0);
        ASCII  : out std_logic_vector(7 downto 0)
    );
end entity;

architecture Behavioral of HEX2Ascii is
begin
```

```
    process (HEX) is
    begin
        case HEX is
        when "0000" => ASCII <= X"30"; -- 0
        when "0001" => ASCII <= X"31"; -- 1
        when "0010" => ASCII <= X"32"; -- 2
        when "0011" => ASCII <= X"33"; -- 3
        when "0100" => ASCII <= X"34"; -- 4
        when "0101" => ASCII <= X"35"; -- 5
        when "0110" => ASCII <= X"36"; -- 6
        when "0111" => ASCII <= X"37"; -- 7
        when "1000" => ASCII <= X"38"; -- 8
        when "1001" => ASCII <= X"39"; -- 9
        when "1010" => ASCII <= X"41"; -- A
        when "1011" => ASCII <= X"42"; -- B
        when "1100" => ASCII <= X"43"; -- C
        when "1101" => ASCII <= X"44"; -- D
        when "1110" => ASCII <= X"45"; -- E
        when "1111" => ASCII <= X"46"; -- F
        when OTHERS => ASCII <= X"0000"; --undefined state
        end case;
    end process;
end architecture;
```

Timer Interrupt SDK C file

```
// Variable for data to send
uint16_t arr[] =
{0x0000,0xFADE,0xCAFE,0x4B1D,0xFEED,0x1BAD,0xD00D,0xDEAD,0xBEEF,0xF00D};
#define arr_size 10
// Variable for memory addresses of the interfaces
#define LCD_AXI 0x43C10000
#define Serial_7Seg_AXI 0x43C00000
*****************************************************************************/

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
int loop_count = 0;
void TimerCounterHandler(void *CallBackRef, u8 TmrCtrNumber)
{
    XTmrCtr *InstancePtr = (XTmrCtr *)CallBackRef;

    /*
     * Check if the timer counter has expired, checking is not necessary
     * since that's the reason this function is executed, this just shows
```

```
     * how the callback reference can be used as a pointer to the instance
     * of the timer counter that expired, increment a shared variable so
     * the main thread of execution can see the timer expired
     */
    if (XTmrCtr_IsExpired(InstancePtr, TmrCtrNumber)) {
        // Print data to Computer Terminal
        xil_printf("\rLCD_DATA[%d] = %04X \n\r", loop_count, arr[loop_count]);

        // Print data to LCD
        Xil_Out32(LCD_AXI, arr[loop_count]);
        // Print data to 7 segment display
        Xil_Out32(Serial_7Seg_AXI, arr[loop_count]);

        loop_count = loop_count + 1;

        if (loop_count == arr_size){
            loop_count = 0;
        }


        if (TimerExpired == 3) {
            XTmrCtr_SetOptions(InstancePtr, TmrCtrNumber, 0);
        }
    }
}

/****************************************************************************/
```