Available online at www.elixirpublishers.com (Elixir International Journal)

Electronics Engineering

Elixir Elec. Engg. 81 (2015) 31999-32004

ISSN: 2229-712X

CPLD-Based Data Acquisition System with High Speed Interface

Dr.Haresh Pandya[1], Mahesh Rangapariya[1] and Jitendra Rajput[2]

[1]Department of Electronics, Saurashtra University, Rajkot- 360005, Gujarat, India.

[2]SSR College of Arts, Commerce & Science (Affiliated to Savitribai Phule Pune University), Silvassa- 396230, D & NH, India.

ARTICLE INFO

Article history:

Received: 2 March 2015;

Received in revised form:

19 April 2015;

Accepted: 27 April 2015;

Keywords

Data Acquisition system,

VHDL (VHSIC Hardware Description Language),

CPLD (Complex Programmable Logic Devices),

ADC0808,

LCD (Liquid Crystal Display).

ABSTRACT

This paper presents a novel approach to the design and implementation of CPLD (Complex Programmable Logic Devices) based DAS (Data Acquisition System) for varies application. This technique performs the acquisition of physical signal, conversion of analog signal to digital signal and storing of the information. The core heart of the proposed system is CPLD, which allows individual modules on a chip to work independently from each other which is configured and programmed to acquire real time data. The data for the process is acquired using suitable temperature and gas sensors. Signal conditioners are designed for each sensor and are tested in real time. The ADC0808 (analog to digital converter) is adopted for this system, which is a high speed monolithic CMOS device with an 8-bit, 8-channel analog-to-digital converter using successive approximation as the conversion technique. Cool Runner-II CPLD by Xilinx is used as the main controller from which all modules are implemented in VHDL using Xilinx ISE Design Suite9.2 and simulated using Isim.

Introduction

Data acquisition is the process by which physical phenomena from the real world are transformed into electrical signals that are measured and converted into a digital format for processing, analysis, and storage. The required data can be collected from any peripheral input devices, such as transducers, sensors and other subsystems. In data acquisition system, it is a growing challenge to acquire the data at a required rate and to accumulate the data in an on chip memory processor. There are devices like microprocessors; microcontrollers and DSP are available which can be programmed as a data acquisition system. The main disadvantage of using these devices is their slower data acquisition speed, non-availability of sufficient on-chip memory. Apart from this, the rigidity in the hardware configuration of these devices does not allow flexibility for the user in configuring these devices according to the requirement.

To overcome these drawbacks this research work proposes a novel technique of design and develop a data acquisition system using CPLD which provides flexibility in configuring the device according to the user requirement. The major defining characteristic of the CPLD is that it can be reprogrammed. Programming a CPLD is very different from a microprocessor or a DSP processor. Microprocessor is a stored program computer. A computer system contains both a CPU and a separate memory that stores the instruction and data. CPLD program is interwoven into the structure of CPLD. A CPLD does not fetch instructions. The CPLD's programming directly implements the logic functions and interconnections. In the CPLD's there is no wait for completing the design to obtain a working chip. The design can be programmed into the chip and can be tested immediately. When a CPLD is used in final design, the jump from prototype to product is much smaller and easier. They are having a large number of input and output lines compared to microprocessors, microcontrollers and DSP's. CPLD's are having a higher processing speed compared to microprocessors and microcontrollers. With CPLD devices, it is possible to tailor the design to fit the requirements of applications.

The data acquisition system is broadly utilized in a number of automatic test and measuring equipments. They can be used to collect the required data from any peripheral input devices, such as meters, sensors and etc. via controlling software. The measured data could be stored in the PC. Their values can be shown numerically whereas their relationship can be displayed graphically as a curve on the screen and value can be display on LCD screen. This paper proposed a design of the data acquisition system using CPLD, interfacing to a PC. The system has capability to receive the digital signals from multi-channels sensors with eight different ADC channel.

Overall System

The design mainly involves the development of signal conditioning circuits for varies sensors used in the application and programming the CPLD using VHDL. It presents the ADC connection to the eight different (analog to digital converter) sensors with eight different Parallel buses. CPLD utilized as a data acquisition system is programmed to fetch the data at the output of ADC with the help of a multiplexer and the digital data is stored in its internal RAM. The CPLD collects the individual data from all ADC sensors. After that it produces a stream of data through the output USB port, which sends these ADC data to the LCD. We have written a specific application program to control the LCD. This program has a function to communicate to the CPLD so that the PC could prepare itself for the data transfer. The block diagram of CPLD based data acquisition system is given below in Fig. 1.

Firstly, the PC will check the CPLD for data availability on the system. The analog inputs corresponding to temperature and gas are given to signal conditioning circuit which consists of a multiplexer. The multiplexed analog signals are taken as analog

Tele:

E-mail addresses: jeetusingh86@yahoo.co.in

input of ADC and corresponding 8 bit digital data is obtained. Here ADC0808 works in free running mode. So 8 bit digital data is directly obtained at CPLD ports and is stored in the Block RAM. The stored data is then compared with threshold values already stored in the CPLD. First the data corresponding to temperature and then gas is considered. After comparison when the stored data corresponding to either temperature or gas is greater than threshold value, it indicates that there is chance for fire to occur which is shown by using an alarm at the output. The data will be interpreted into separate data bytes for the individual channels. Finally, the data can be shown to the user, and saved to the main database at the same time. The hardware consists of temperature sensor LM35, Gas sensor MQ6 and their signal conditioning circuits, IC555 Timer, ADC 0808 and Cool Runner-II CPLD. After that it will send a set of the instructions to the CPLD for getting these data from USB port.

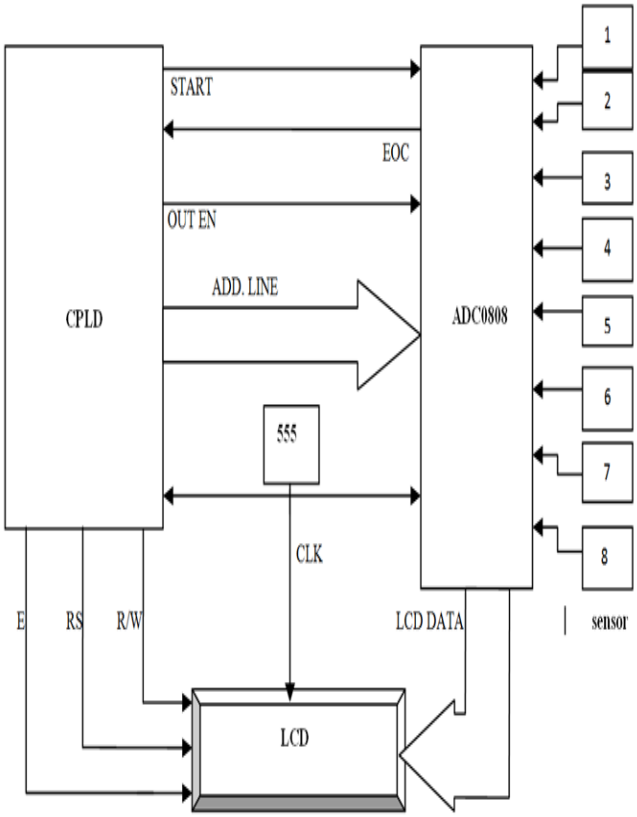


Fig.1. Block diagram of experimental setup for Data Acquisition System

Cool Runner-II CPLD

The processing unit Cool Runner-II CPLD Starter Kit from Xilinx Company is employed for this design. The Cool Runner-II Evaluation Board is a complete USB-powered circuit development platform for the Xilinx Cool Runner-II CPLD. The board includes highly efficient power supplies, a user configurable oscillator, several user I/O devices, a real-time current meter, and a USB port for board power and CPLD programming. The board includes five expansion connectors that route 64 signals available from the CPLD to external circuits to expand board capability. Board features include A 256 microcells Cool Runner-II CPLD in a TQ-144 package. An on-board USB port for board power, CPLD programming and user data transfers. An on-board three-channel 16-bit A/D converter that measures real-time current on VCCINT and the two VCCIO banks during board operation (data is sent to the PC for display via the USB cable). A user-configurable silicon oscillator (1000/100/10 kHz) and socket for a second crystal oscillator. 64 I/O signals available on expansion connectors (32 on Pmod connectors; 32 on a parallel connector).

B. ADC0808

The ADC0808 data acquisition component is a monolithic CMOS device with an 8-bit analog-to-digital converter, 8-channel multiplexer and microprocessor compatible control logic. The 8-bit A/D converter uses successive approximation as the conversion technique. The converter features a high impedance chopper stabilized comparator, a 256R voltage divider with analog switch tree and a successive approximation register. The 8-channel multiplexer can directly access any of 8-single-ended analog signals. The device eliminates the need for external zero and full-scale adjustments. Easy interfacing to microprocessors is provided by the latched and decoded multiplexer address inputs and latched TTL TRI-STATE outputs in this work ADC0804 operates in free running mode which provides self-clocking and so no control signal is needed for ADC from CPLD.

IC555 TIMER

These devices are precision timing circuits capable of producing accurate time delays or oscillation. In the astable mode of operation, the frequency and duty cycle can be controlled independently with two external resistors and a single external capacitor. The threshold and trigger levels normally are two-thirds and one-third, respectively, of VCC. These levels can be altered by use of the control-voltage terminal. The output circuit is capable of sinking or sourcing current up to 200 mA. Operation is specified for supplies of 5 V to 15 V. With a 5-V supply, output levels are compatible with TTL inputs. The NE555 is characterized for operation from 0°C to 70°C. All above process require external master clk pulses. This is provided by IC555 timer with 3.3 operating voltage.

Liquid Crystal Display (LCD)

LCD is used to display three values, one is the system initialising message, second is the input data entered through keyboard, third one is the selected channel respective sensor output reading. The 8X2 LCD is used, which contains two lines and 8 characters in each line. It is a general purpose alphanumeric display.

Parallel Protocol

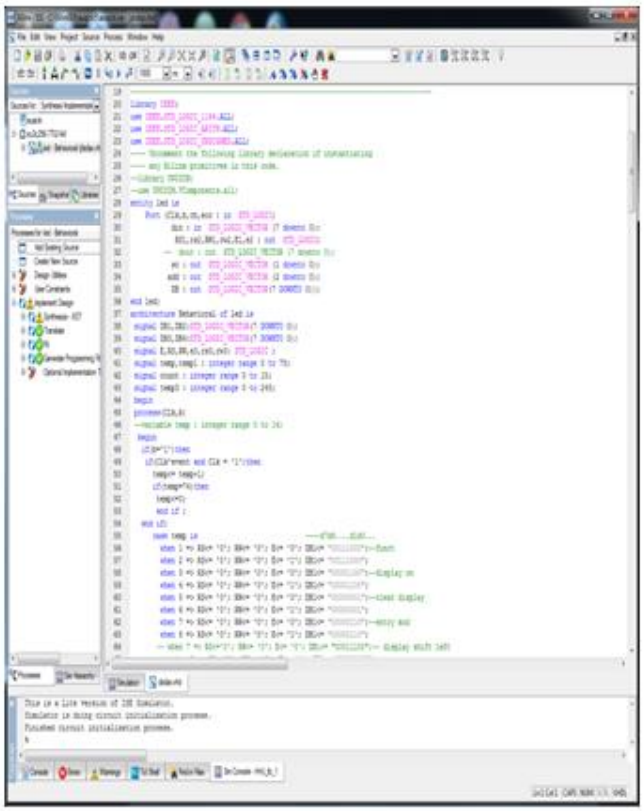
This protocol is the traditional type for most ADC's. It has the advantage of the high speed throughput. This design uses ADC0808 for the peripheral device. Figure 3 presents the simulation of how the CPLD gets data from this ADC. There are two main steps in the conversion process:

- The CPLD sends the start signal to activate the ADC then it will wait for the acknowledge signal.
- After finishing the data converting, the ADC will send the acknowledge signal to the CPLD. Then the CPLD reads the data from the bus. After that the CPLD sends the start signal to activate ADC again for getting the data on next read cycle. Obviously, this data acquisition is sow simple and fast. Thus, this protocol should be employed with the high speed system.

VHDL Design Process

There has to be a better way, and, of course, there is. It is called high-level design (HLD), behavioural, or hardware description language (HDL). For our purposes, these three terms are essentially the same thing. The idea is to use a high-level language to describe the circuit in a text file rather than a graphical low-level gate description. The term behavioural is used because in this powerful language you describe the function or behaviour of the circuit in words rather than figuring out the appropriate gates needed to create the application. There are two major flavours of HDL: VHDL and Verilog.

VHDL Program Window



VHDL Program Code for Data Acquisition System

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity led is
    Port (Clk,b,ch,eoc : in  STD_LOGIC;
          din : in  STD_LOGIC_VECTOR (7 downto 0);
           RS1,rs2,RW1,rw2,E1,e2 : out  STD_LOGIC;
            sc : out  STD_LOGIC_VECTOR (1 downto 0);
          add : out  STD_LOGIC_VECTOR (2 downto 0);
          DB : out  STD_LOGIC_VECTOR(7 DOWNTO 0));
end led;

architecture Behavioral of led is
   signal DB1,DB2:STD_LOGIC_VECTOR(7 DOWNTO 0);
   signal DB3,DB4:STD_LOGIC_VECTOR(7 DOWNTO 0);
   signal E,RS,RW,e3,rs3,rw3: STD_LOGIC ;
   signal temp,temp1 : integer range 0 to 75;
   signal count : integer range 0 to 15;
   signal temp3 : integer range 0 to 245;
begin
process(Clk,b)
   begin
    if(b='1')then
     if(Clk'event and Clk = '1')then  temp<= temp+1;
      if(temp=74)then temp<=0;
    end if ;
end if;
     case temp is                          ----d7d6....d1d0...
       when 1 => RS<= '0'; RW<= '0'; E<= '0'; DB1<=
"00111000";--funct
       when 2 => RS<= '0'; RW<= '0'; E<= '1'; DB1<=
"00111000";
       when 3 => RS<= '0'; RW<= '0'; E<= '0'; DB1<=
"00001100";--display on
       when 4 => RS<= '0'; RW<= '0'; E<= '1'; DB1<=
"00001100";
       when 5 => RS<= '0'; RW<= '0'; E<= '0'; DB1<=
"00000001";--clear display
       when 6 => RS<= '0'; RW<= '0'; E<= '1'; DB1<=
"00000001";
       when 7 => RS<= '0'; RW<= '0'; E<= '0'; DB1<=
"00000110";--entry mod
       when 8 => RS<= '0'; RW<= '0'; E<= '1'; DB1<=
"00000110";
       when 9 =>RS<= '1'; RW<= '0'; E<= '0'; DB1<=
"01010111"; --W
       when 10=>RS<= '1'; RW<= '0'; E<= '1'; DB1<=
"01010111";
       when 11=>RS<= '1'; RW<= '0'; E<= '0'; DB1<=
"01000101"; --E
       when 12=>RS<= '1'; RW<= '0'; E<= '1'; DB1<=
"01000101";
       when 13=>RS<= '1'; RW<= '0'; E<= '0'; DB1<=
"01001100"; --L
       when 14=>RS<= '1'; RW<= '0'; E<= '1'; DB1<=
"01001100";
       when 15=>RS<= '1'; RW<= '0'; E<= '0'; DB1<=
"10110000"; -- -
       when 16=>RS<= '1'; RW<= '0'; E<= '1'; DB1<=
"10110000";
       when 17=>RS<= '1'; RW<= '0'; E<= '0'; DB1<=
"01000011"; -- C
       when 18=>RS<= '1'; RW<= '0'; E<= '1'; DB1<=
"01000011";
       when 19=>RS<= '1'; RW<= '0'; E<= '0'; DB1<=
"01001111"; -- O
       when 20=>RS<= '1'; RW<= '0'; E<= '1'; DB1<=
"01001111";
       when 21=>RS<= '1'; RW<= '0'; E<= '0'; DB1<=
"01001101"; -- M
       when 22=>RS<= '1'; RW<= '0'; E<= '1'; DB1<=
"01001101";
       when 23=>RS<= '1'; RW<= '0'; E<= '0'; DB1<=
"01000101"; -- E
       when 24=>RS<= '1'; RW<= '0'; E<= '1'; DB1<=
"01000101";
       when 25=>RS<= '1'; RW<= '0'; E<= '0'; DB1<=
"10100000"; -- space
       when 26=>RS<= '1'; RW<= '0'; E<= '1'; DB1<=
"10100000";
       when 27=>RS<= '1'; RW<= '0'; E<= '0'; DB1<=
"01010100"; -- T
       when 28=>RS<= '1'; RW<= '0'; E<= '1'; DB1<=
"01010100";
       when 29=>RS<= '1'; RW<= '0'; E<= '0'; DB1<=
"01001111"; -- O
       when 30=>RS<= '1'; RW<= '0'; E<= '1'; DB1<=
"01001111";
       when 31=>RS<= '1'; RW<= '0'; E<= '0'; DB1<=
"10100000"; -- space
       when 32=>RS<= '1'; RW<= '0'; E<= '1'; DB1<=
"10100000";
       when 33=>RS<= '1'; RW<= '0'; E<= '0'; DB1<=
"01000011"; -- C
       when 34=>RS<= '1'; RW<= '0'; E<= '1'; DB1<=
"01000011";
```

```
      when 35=>RS<= '1'; RW<= '0'; E<= '0'; DB1<= "01010000"; -- P
      when 36=>RS<= '1'; RW<= '0'; E<= '1'; DB1<= "01010000";
      when 37=>RS<= '1'; RW<= '0'; E<= '0'; DB1<= "01001100"; -- L
      when 38=>RS<= '1'; RW<= '0'; E<= '1'; DB1<= "01001100";
      when 39=>RS<= '1'; RW<= '0'; E<= '0'; DB1<= "01000100"; -- D
      when 40=>RS<= '1'; RW<= '0'; E<= '1'; DB1<= "01000100";
      when 41=>RS<= '0'; RW<= '0'; E<= '0'; DB1<= "11000001"; --give 2-line--address
      when 42=>RS<= '0'; RW<= '0'; E<= '1'; DB1<= "11000001";
      when 43=>RS<= '1'; RW<= '0'; E<= '0'; DB1<= "01000001"; -- A
      when 44=>RS<= '1'; RW<= '0'; E<= '1'; DB1<= "01000001";
      when 45=>RS<= '1'; RW<= '0'; E<= '0'; DB1<= "01000100"; -- D
      when 46=>RS<= '1'; RW<= '0'; E<= '1'; DB1<= "01000100";
      when 47=>RS<= '1'; RW<= '0'; E<= '0'; DB1<= "01000011"; -- C
      when 48=>RS<= '1'; RW<= '0'; E<= '1'; DB1<= "01000011";
      when 49=>RS<= '1'; RW<= '0'; E<= '0'; DB1<= "00110000"; -- 0
      when 50=>RS<= '1'; RW<= '0'; E<= '1'; DB1<= "00110000";
      when 51=>RS<= '1'; RW<= '0'; E<= '0'; DB1<= "00111000"; -- 8
      when 52=>RS<= '1'; RW<= '0'; E<= '1'; DB1<= "00111000";
      when 53=>RS<= '1'; RW<= '0'; E<= '0'; DB1<= "10100000"; -- space
      when 54=>RS<= '1'; RW<= '0'; E<= '1'; DB1<= "10100000";
      when 55=>RS<= '1'; RW<= '0'; E<= '0'; DB1<= "01000011"; -- C
      when 56=>RS<= '1'; RW<= '0'; E<= '1'; DB1<= "01000011";
      when 57=>RS<= '1'; RW<= '0'; E<= '0'; DB1<= "01001111"; -- O
      when 58=>RS<= '1'; RW<= '0'; E<= '1'; DB1<= "01001111";
      when 59=>RS<= '1'; RW<= '0'; E<= '0'; DB1<= "01001110"; -- N
      when 60=>RS<= '1'; RW<= '0'; E<= '1'; DB1<= "01001110";
      when 61=>RS<= '1'; RW<= '0'; E<= '0'; DB1<= "01010100"; -- T
      when 62=>RS<= '1'; RW<= '0'; E<= '1'; DB1<= "01010100";
      when 63=>RS<= '1'; RW<= '0'; E<= '0'; DB1<= "01010010"; -- R
      when 64=>RS<= '1'; RW<= '0'; E<= '1'; DB1<= "01010010";
      when 65=>RS<= '1'; RW<= '0'; E<= '0'; DB1<= "01001111"; -- O
      when 66=>RS<= '1'; RW<= '0'; E<= '1'; DB1<= "01001111";
      when 67=>RS<= '1'; RW<= '0'; E<= '0'; DB1<= "01001100"; -- L
      when 68=>RS<= '1'; RW<= '0'; E<= '1'; DB1<= "01001100";
      when 69=>RS<= '0'; RW<= '0'; E<= '0'; DB1<= "00011100";-- display shift right
      when 70=>RS<= '0'; RW<= '0'; E<= '1'; DB1<= "00011100";
      when 71=>RS<= '0'; RW<= '0'; E<= '0'; DB1<= "00011000";-- display shift left
      when 72=>RS<= '0'; RW<= '0'; E<= '1'; DB1<= "00011000";
      when 73=>RS<= '0'; RW<= '0'; E<= '0'; DB1<= "00011000";-- display shift left
      when 74=>RS<= '0'; RW<= '0'; E<= '1'; DB1<= "00011000";
    -- when 77=>RS<= '0'; RW<= '0'; E<= '0'; DB1<= "00011000";-- display shift left
   -- when 78=>RS<= '0'; RW<= '0'; E<= '1'; DB1<= "00011000";
      when others =>null ;
   end case;
        DB<=DB1 ; E1<=E ; RS1<=RS ; RW1<= RW ; e2<=E ; rs2<=RS ; rw2<= RW ;
  end if ;
if(b='0')then
   if(Clk'event and Clk = '1')then temp1<= temp1+1;
      if(temp1=43)then temp1<=0;
     end if ;
   end if;
case temp1 is                    ----d7d6....d1d0...
when 1=>rs3<= '0'; rw3<= '0';  e3<= '0'; DB2<= "00000001";-- clear display
when 2=>rs3<= '0'; rw3<= '0';  e3<= '1'; DB2<= "00000001";
when 3=>rs3<= '0'; rw3<= '0';  e3<= '0'; DB2<= "00000110";-- entry mod
when 4=>rs3<= '0'; rw3<= '0';  e3<= '1'; DB2<= "00000110";
when 5=>rs3<= '0'; rw3<= '0';  e3<= '0'; DB2<= "10000010"; -- give 1-line--address
when 6=>rs3<= '0'; rw3<= '0';  e3<= '1'; DB2<= "10000010";
when 7=>rs3<= '1'; rw3<= '0';  e3<= '0'; DB2<= "01001001";  -- I
when 8=>rs3<= '1'; rw3<= '0';  e3<= '1'; DB2<= "01001001";
when 9=>rs3<= '1'; rw3<= '0';  e3<= '0'; DB2<= "01001110";  -- N
when 10=>rs3<= '1'; rw3<= '0';  e3<= '1'; DB2<= "01001110";
when 11=>rs3<= '1'; rw3<= '0';  e3<= '0'; DB2<= "01010000"; -- P
when 12=>rs3<= '1'; rw3<= '0';  e3<= '1'; DB2<= "01010000";
when 13=>rs3<= '1'; rw3<= '0';  e3<= '0'; DB2<= "01010101"; -- U
when 14=>rs3<= '1'; rw3<= '0';  e3<= '1'; DB2<= "01010101";
when 15=>rs3<= '1'; rw3<= '0';  e3<= '0'; DB2<= "01010100"; -- T
when 16=>rs3<= '1'; rw3<= '0';  e3<= '1'; DB2<= "01010100";
when 17=>rs3<= '1'; rw3<= '0';  e3<= '0'; DB2<= "10100000"; -- space
when 18=>rs3<= '1'; rw3<= '0';  e3<= '1'; DB2<= "10100000";
when 19=>rs3<= '1'; rw3<= '0';  e3<= '0'; DB2<= "01010110"; -- V
when 20=>rs3<= '1'; rw3<= '0';  e3<= '1'; DB2<= "01010110";
when 21=>rs3<= '1'; rw3<= '0';  e3<= '0'; DB2<= "01001111"; -- O
when 22=>rs3<= '1'; rw3<= '0';  e3<= '1'; DB2<= "01001111";
```

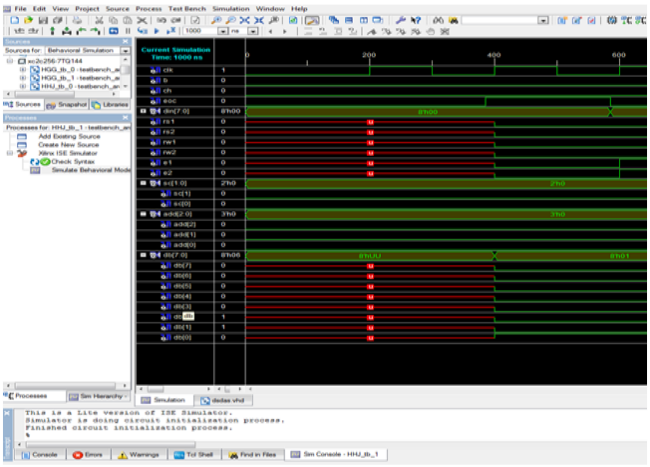
```
when 23=>rs3<= '1'; rw3<= '0';  e3<= '0'; DB2<= "01001100"; --
L
when 24=>rs3<= '1'; rw3<= '0';  e3<= '1'; DB2<= "01001100";
when 25=>rs3<= '1'; rw3<= '0';  e3<= '0'; DB2<= "01010100"; --
T
when 26=>rs3<= '1'; rw3<= '0';  e3<= '1'; DB2<= "01010100";
when 27=>rs3<= '1'; rw3<= '0';  e3<= '0'; DB2<= "01000001"; --
A
when 28=>rs3<= '1'; rw3<= '0';  e3<= '1'; DB2<= "01000001";
when 29=>rs3<= '1'; rw3<= '0';  e3<= '0'; DB2<= "01000111"; --
G
when 30=>rs3<= '1'; rw3<= '0';  e3<= '1'; DB2<= "01000111";
when 31=>rs3<= '1'; rw3<= '0';  e3<= '0'; DB2<= "01000101"; --
E
when 32=>rs3<= '1'; rw3<= '0';  e3<= '1'; DB2<= "01000101";
when 33=>rs3<= '0'; rw3<= '0'; e3<= '0'; DB2<= "11000110"; --
give 2-line--address
when 34=>rs3<= '0'; rw3<= '0'; e3<= '1'; DB2<= "11000110";
when 35=>rs3<= '1'; rw3<= '0'; e3<= '0'; DB2<= DB3;
when 36=>rs3<= '1'; rw3<= '0'; e3<= '1'; DB2<= DB3;
when 37=>rs3<= '1'; rw3<= '0'; e3<= '0'; DB2<= "00101110";
when 38 =>rs3<= '1'; rw3<= '0'; e3<= '1'; DB2<= "00101110";  --
.
when 39 =>rs3<= '1'; rw3<= '0'; e3<= '0'; DB2<= DB4;
when 40 =>rs3<= '1'; rw3<= '0'; e3<= '1'; DB2<= DB4;
when 41=>rs3<= '1'; rw3<= '0';  e3<= '0'; DB2<= "01010110"; --
V
when 42=>rs3<= '1'; rw3<= '0';  e3<= '1'; DB2<= "01010110";
when others =>null ;
        end case;
        DB<=DB2 ;E1<=e3 ; RS1<=rs3 ; RW1<= rw3 ;e2<=e3 ;
rs2<=rs3 ; rw2<= rw3 ;
     end if ;
end process;
process(Clk)
   begin
if(Clk'event and Clk = '0')then  temp3 <= temp3+1;
      if(temp3=225)then temp3<=0;
   end if;
end if;
end process;
    with temp3 select
           sc<= "11" when  220,---up to 210 allow not bellow
that.
                 "11" when  221,"11" when  222,"11" when
223,"00" when others;
process (eoc,din)
   begin
   if(eoc='1')then  --d7--d0
if((din>="00000000")  and  (din<=  "00010010"))then
DB3<="00110000"; DB4<="00110011";      --0.3V
elsif((din>="00010011")  and  (din<=  "00010100"))then
DB3<="00110000"; DB4<="00110100";   --0.4V
elsif((din>="00010101")  and  (din<=  "00101000"))then
DB3<="00110000"; DB4<="00110101";   --0.5V
elsif((din>="00101001")  and  (din<=  "00101100"))then
DB3<="00110000"; DB4<="00110110";   --0.6V
elsif((din>="00101101")  and  (din<=  "00110000"))then
DB3<="00110000"; DB4<="00110111";   --0.7V
elsif((din>="00110001")  and  (din<=  "00111000"))then
DB3<="00110000"; DB4<="00111000";   --0.8V
elsif((din>="00111001")  and  (din<=  "01000000"))then
DB3<="00110000"; DB4<="00111001";   --0.9V
elsif((din>="01000001")  and  (din<=  "01001000"))then
DB3<="00110001"; DB4<="00110000";   --1.0V
elsif((din>="01001001")  and  (din<=  "01010000"))then
DB3<="00110001"; DB4<="00110001";   --1.1V
elsif((din>="01010001")  and  (din<=  "01010110"))then
DB3<="00110001"; DB4<="00110010";   --1.2V
elsif((din>="01010111")  and  (din<=  "01011110"))then
DB3<="00110001"; DB4<="00110011";   --1.3V
elsif((din>="01011111")  and  (din<=  "01100110"))then
DB3<="00110001"; DB4<="00110100";   --1.4V
elsif((din>="01100111")  and  (din<=  "01101110"))then
DB3<="00110001"; DB4<="00110101";   --1.5V
elsif((din>="01101111")  and  (din<=  "01110100"))then
DB3<="00110001"; DB4<="00110110";   --1.6V
elsif((din>="01110101")  and  (din<=  "01111100"))then
DB3<="00110001"; DB4<="00110111";   --1.7V
elsif((din>="01111101")  and  (din<=  "10000100"))then
DB3<="00110001"; DB4<="00111000";   --1.8V
elsif((din>="10000101")  and  (din<=  "10001100"))then
DB3<="00110001"; DB4<="00111001";   --1.9V
elsif((din>="10001101")  and  (din<=  "10010010"))then
DB3<="00110010"; DB4<="00110000";   --2.0V
elsif((din>="10010011")  and  (din<=  "10100000"))then
DB3<="00110010"; DB4<="00110010";   --2.2V
elsif((din>="10100001")  and  (din<=  "10101110"))then
DB3<="00110010"; DB4<="00110100";   --2.4V
elsif((din>="10101111")  and  (din<=  "10111110"))then
DB3<="00110010"; DB4<="00110110";   --2.6V
elsif((din>="10111111")  and  (din<=  "11001100"))then
DB3<="00110010"; DB4<="00111001";   --2.8V
elsif((din>="11001101")  and  (din<=  "11011100"))then
DB3<="00110011"; DB4<="00110000";   --3.0V
elsif((din>="11011101")  and  (din<=  "11111111"))then
DB3<="00110011"; DB4<="00110100";   --3.4V
        end if ;
     end if;
end process;
process (ch)
   begin
      if(ch'event and ch= '0')then count<= count+1;
        if(count>15)then count<=0;
          end if;
        end if;
      case count is
when 0 => add<= "000"; when 1 => add<= "000"; when 2   =>
add<= "001"; when 3 => add<= "001"; when 4 => add<= "010";
when 5 => add<= "010"; when 6   => add<= "011"; when 7 =>
add<= "011";
when 8 => add<= "100"; when 9 => add<= "100"; when 10 =>
add<= "101"; when 11 => add<= "101";
when 12 => add<= "110"; when 13 => add<= "110"; when 14
=> add<= "111";

when 15 => add<= "111";
      end case ;
  end process;
end Behavioral;
```

Simulate Behavioral Model

You can now run a functional simulation on the display_drive module. With display_drive.vhd highlighted in the **Source** window, the **Process** window will give all the available operations for the particular module. A VHDL file can be synthesised and then implemented through to a bit stream. Normally, a design consists of several lower level modules

wired together by a top-level file. In this instance, we are going to simulate only one lower-level module to introduce the functional simulation methodology.



Conclusions

The proposed project-based approach encompasses whole engineering cycle, starting from specification, through design, modelling, simulation and verification, implementation, to performance measurement, and closing the cycle with the design improvements in order to maximize performance at minimal cost. Students have an opportunity to apply their theoretical knowledge of hardware description languages, digital design and computer architecture, and to gain real-world experience in developing IP cores. The work in small teams follows a real industry pattern, with one student designated as team leader, and instructor conducting design reviews every week. We feel that such projects are essential to educate future architects of complex systems-on-a-chip.

References

- Data Acquisition Linear devices Data Book - National Semiconductor.
- VHDL Programming by Example By Douglas L. Perry Circuit Design with VHDL By Volnei A. Pedroni
- DIDITAL DESIGN principles & practices By JOHN F. WAKERLY
- Fundamentals of Digital Logic with VHDL Design. By Stephen Brown and Zonked Varanasi
- The Practical Xilinx Designer Lab Book- Prentice Hall. By David Van den Bout
- The Practical Xilinx Designer Lab Book- Prentice Hall. By David Van den Bout
- VHDL Starter's Guide.-Prentice Hall. By Sudhakar Yalamanchili
- Digital Designing with Programmable Logic Devices.-Prentice Hall. By John W. Carter
- International Journal of Advanced Research in Computer Science and Software Engineering (Volume 3, Issue 8, August 2013)
- Programmable Logic Design Quick Start Guide (UG500 (v1.0) May 8, 2008)