Interfacing Force Sensor to on Board ADC of Spartan3E

Mustafa Salim
University of Mosul
Iraq

Rafid Ahmed
University of Mosul
Iraq

ABSTRACT

Force sensor from interlink that use square pad for force sensing is useful for many applications, as the resolution of this sensor continues and FPGA is digital, an ADC is required to deal with such sensor and as the embedded ADC of Spartan 3E kit need complex signals synchronization to work. This paper present a hardware design and optimized implementation based VHDL coding of control unit that make the ADC of Spartan 3E kit operate in real time. The system use Moore states machine style for generating ADC synchronization signals. Also, the result observed via new approach, which is instead of using chip scope for displaying values of ADC, LabVIEW FPGA environment is used so it's possible to display result on PC via USB. The control is optimized and efficient due to using only 1% of FPGA chip and the measured force can be displayed in real time and the proposed control unit can be work as standalone or coprocessor architecture.

General Terms

FPGA, VHDL, LabVIEW FPGA, Force, Sensor.

Keywords

FPGA, Analog to Digital Convertor, Spartan 3E FPGA, Force Sensor, FSR, Programmable Pre-Amplifier, LTC 1407A, ADC, Labview FPGA.

1. INTRODUCTION

Force sensing resistor from Interlink is the most versatile force sensing technology on the market today. These innovative sensors provide a response in resistance with an increase in applied force. These sensors provides engineers and designers with a durable, reliable, easy to measure and low cost solution for human machine interface touch [1]. The FPGA [2] with reconfigurable logic feature, portability, low consumption of energy, high speed and large memory for data storage are great for use for development of complex applications. FPGA configuration based VHDL [3] coding allow us to develop specific hardware architectures with maximum flexibility. This specific feature of the VHDL that give high degree of design anatomy comparing to others implementation methods. LabVIEW FPGA [4] is new hardware design tool that can be used to design different systems, LabVIEW FPGA provide different features such easy hardware component synchronization and interfacing between PC and Spartan 3E FPGA chip. In the literature several control units of embedded ADC proposed. In [5] Istiyanto J. E present a design of a controller based states machine using VHDL for the amplifier and the embedded ADC and tested it using a function generator and an oscilloscope and the chip utilization was 3%. Also in [6] G. Crouch from National Instruments provide a control unit of ADC with signal filtering based on pure LabVIEW FPGA environment and display the filtered result on PC using USB.

The disadvantage of approach is that the chip utilization report based on our LabVIEW FPGA in the lab for (ADC unit only) consumed about 25%. In [7] S. Chaurasiya presents an interface unit of the embedded ADC and the results were observed via ChipScope-Pro and LEDs also they provide VHDL code base on states machine for such unit and our FPGA synthesis report of code indicate that the chip utilization about 4%. In this paper, a hardware design and implementation based VHDL coding of control unit is presented that able to synchronize signals of embedded ADC of Spartan 3E with displaying the result on PC via LabVIEW FPGA environment that provide direct method for connecting between chip and PC via USB.2 with only 1% FPGA chip utilization. This paper is divided into three sections: section two describes the force sensor operation, section three shows the hardware implementation and how each ADC signals synchronized with result, section four presents the conclusion, five and six give reference and VHDL code of the control unit respectively.

2. FORCE SENSOR OPERATION

The basic force sensing resistors consists of two membranes separated by a thin air. The air gap is maintained by a spacer around the edges and by the rigidity of the two membranes. One of the membranes has two sets of interdigitated fingers that are electrically distinct, with each set connecting to one trace on a tail. The other membrane is coated with FSR ink. Around the perimeter of the sensor is a spacer adhesive that serves both to separate the two substrates and hold the sensor together. The inside surface of one substrate is coated with FSR carbon-based ink. When the two substrates are pressed together, the microscopic protrusions on the FSR ink surface short across the interdigitated fingers of the facing surface [8].

3. H/W IMPLEMENTATION

The block diagram of entire implemented system shown in Figure (1). It consists of force-to-voltage conversion, embedded amplifier, ADC and signals controller base on states machine that consist of two main states and several sub states that are synchronized together to generate specific time signals transaction. The Spartan 3E FPGA includes a two channel analog capture that is consisting of a programmable amplifier and ADC. The analog circuit capture and convert the voltage on VINA or VINB on board to a 14-bit digital at the same time under the following [9]:

$$\text{ADC}[13:0] = 8192 * \text{GAIN} * (\text{Vin} - 1.65 \text{ v}) / (1.25 \text{ v}) \dots\dots (1)$$

The gain is the current setting that is loaded into the amplifier. The allowable setting for gain and allowable voltages applied to VINA and VINB inputs appear in table (1) [9]. For example if gain set to -1 then the input voltage range will be with 0.4v as minimum and 2.9 v with maximum. Figure (2) show two examples that both represent the ideal response of the analogue inputs. In order to program the amplifier by using

Fig.1: Complete system [initial graph was taken from [9]]

Table 1. Range selecting

Gain value	A3A2A1A0 B3B2B1B0			
-1	0	0	0	1
-2	0	0	1	0
-5	0	0	1	1

Fig 2: Ideal response of the ADC.

FPGA with selected gain value from table (1); three SPI signals (AMP_CS, SPI_MOSI and SPI_SCK) must be generated under specific timing. The amplifier programming starts when the FPGA asserts AMP_CS signal to Low; the amplifier captures serial data on SPI_MOSI (Master Output Slave Input) on the rising edge of the SPI_SCK clock signal [9]. Figure (3) shows timing diagram of the communication process of the implement control unit with amplifier with sending specific gain [00010001] (send bit by bit) that give maximum input voltage range. For any bit transmission and reception in SPI protocol it relative all to SPI_SCK also, as the control unit based on states machine with two main states and several sub states; it can be seen from figure (3) that the starting main state of the unit is amp_states. The gain of amplifier must be sent as an 8-bits consisting of two 4-bits. The most-significant bit B3 is sent first.

After amplifier gain programming, the next step is to read from ADC and in order to read by using FPGA; there are three basic

SPI signals that must be carefully controlled and generated in specific time. These signals include: SPI_MISO, SPI_SCK and AD_CONV [9]. Figure (4) shows the simulation timing diagram of the communication process of the implement control unit with ADC where the signal SPI_MISO (SPI-Master Input Slave Output) set as '1'. The system only read the VINA channel most significant bit first during first 16 SPI_SCK cycle and ignore VINB. As shown in figure (4) when the AD_CONV signal goes high for limited time then goes low the ADC simultaneously capture (VINA and VINB) at the same time. The next capture and conversion are not presented until the next AD_CONV is goes and low again so the frequency of this signal control the sampling frequency of the embedded ADC. The signal frequency can be controlled through using single NOT gate, comparator and counter that all already available in the proposed control unit and the value that will compared with the counter output value can be changed [refer to VHDL code sampling frequency]. The ADC presents the value of the samples as 14 bits two's complement [9]. The embedded ADC requires at least 34 cycles of SPI_SCK [9] that must be distributed as two 14 bits (significant bit first) for channel VINA and VINB and the remaining 6 cycles are not used. The SPI bus signals are shared by other devices on the same board. The other devices must be disabled [9] and it was disabled as shown in figure (5).

Fig 3: Control unit communication with amplifier

Fig4: Control unit communication with ADC.

Fig5: Disabling the other devices through setting on or off value to specific signals.

After successful control unit design and implementation developing using ISE environment [10], The VHDL code was merging in LabVIEW FPGA environment so the real time ADC acquisition of force sensor can displayed on PC with coprocessor architecture as shown in figure (6). Several different force (voltages) changes over time at sensor pad are captured displayed in real time at chart indicator of LabVIEW software as shown in Figure (7). Only the most 8 significant bits was taken from VINA channel. The chip utilization report shown in table (2).

Fig 6: The adopted hardware test approach

Fig7: Different force pressure at sensor

Table 2: FPGA utilization report.

Logic Utilization	Utilization
Number of Slice Flip Flops	1%
Number of 4 input LUTs	1%
Number of occupied Slices	1%

4. CONCLUSION

This paper present a hardware implementation based VHDL of controller unit that make the embedded ADC of Spartan 3E kit operate. The design of amplifier and controller based on Moore states machine and due to that the unit is optimized as it using only 1% from silicon area of the chip. The adopted on chip verification with coprocessor architecture that required merging of VHDL with LabVIEW FPGA contribute in reducing the complexity associated with traditional on chip verification like the chip scope also, as have been noted that the on bored ADC of Spartan 3E kit very affected by the noise and only the most significant bits can be used. Our future works are to connect force sensor to robot arm end effector and developing a control unit for embedded Spartan 3E digital analog converter with proposing embedded real time digital signal processing algorithms such adaptive filter in FPGA.

5. REFERENCES

- [1] Online article at <http://www.interlinkelectronics.com/products.php>. I. Kuon, R. Tessier, and J. Rose "FPGA Architecture: Survey and Challenges", now Publishers Inc., 2008.
- [2] National Instruments Co, "Cycle-Accurate Co-Simulation with Mentor Graphics ModelSim", white papers, Nov 09, 2012.
- [3] V. A. Pedroni "Circuit Design with VHDL ", MIT Press, 2004.
- [4] Online article at <https://www.ni.com/fpga>.
- [5] Istiyanto J.E, "A VHDL-based ADC on FPGA", IEEE International Conference on Instrumentation, Communications, Information Technology, and Biomedical Engineering, Bandung, 23-25 Nov. 2009, pp. 1 -3.

- [6] G. Crouch "National Instruments LabVIEW FPGA Training for the Spartan-3E XUP or NI R-Series Hardware", May 08, 2013.
- [7] ShobhitChaurasia "IMPLEMENTATION OF THE ONBOARD ADC ON SPARTAN 3E FPGA PLATFORM "Department of Computer Science and Engineering, IIT ,Guwahati, Jun 2012.
- [8] Interlink Electronics, "Force Sensing Resistors", integration guide.
- [9] Xilinx Co. "Spartan-3E FPGAS Starter Kit Board User Guide", UG230 (v1.2) January 20, 2011.
- [10] Xilinx Co. "ISE Synthesis and Simulation Design Guide", UG626 (v 14.1) May 8, 2012.
- [11] K. Chapman, "Picoblaze: amplifier and A/D Converter Control for Spartan-3E Starter Kit", 2006.

6. APPENDIX

VHDL code

Here is our VHDL code that can display the captured voltage at 8 bits LEDs of onBoard Spartan 3E.

--- 30/6/2013

library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.NUMERIC_STD.ALL;

use IEEE.std_logic_arith.all;

entity spartan3e_adc is

port (

clk : in std_logic; --- 50MHZ default spartan 3e clock

SPI_MISO : in std_logic; --- read from ADC

enable : in std_logic; ---

SPI_MOSI : out std_logic; --- for gain value

SPI_SCK : out std_logic; --- write/read values during edges of this signal

AD_CONV : out std_logic; --- control the sampling rate of ADC

AMP_CS : out std_logic; --- must be zero during gain set only

leds : out std_logic_vector(7 downto 0);

SPI_SS_B : out std_logic;

SF_CE0 : out std_logic;

FPGA_INIT_B : out std_logic;

DAC_CS : out std_logic;

SF_OE : out std_logic;

SF_WE : out std_logic);

end spartan3e_adc;

architecture Behavioral of spartan3e_adc is

signal en_x : std_logic := '0';

-----states machine declaration-----

type state_type is (amp_states, adc_states);

type amp_state_type is (amp_s1, amp_s2, amp_s3, amp_s4); --

amplifier sub states

type adc_state_type is (adc_s1, adc_s2, adc_s3, adc_s4); --

ADC sub states

signal state : state_type := amp_states;

signal amp_state : amp_state_type := amp_s1;

signal adc_state : adc_state_type := adc_s1;

-----

signal vina_inedx : integer range 0 to 16; --- index for incoming ADC serial data.

begin

SPI_SS_B <= '1';

SF_CE0 <= '1';

FPGA_INIT_B <= '0';

DAC_CS <= '1';

sf_OE <= '1';

sf_WE <= '1';

-----Sampling frequency-----

process (clk)

variable counter: integer range 0 to 30:=0;

begin

if(rising_edge(clk)) then

if(counter = 30) then --- the value of this comparator

control the ADC

en <= en xor '1'; --- reverse one to zero and verse.

counter := 0;

else

counter := counter + 1;

end if;

end if;

end process;

-----

process(en)

variable vina : std_logic_vector(13 downto 0);

variable gain_index : integer range 0 to 7;

constant gain : std_logic_vector(7 downto 0):=

"00010001"; --most significant send first.

variable delay : integer range 0 to 10:=0;

variable vina_inedx : integer range 0 to 17;

variable counter : integer range 0 to 34;

begin

if (rising_edge(en)) then

case state is

-----amp_state1-----

when amp_states =>

case amp_state is

when amp_s1 =>

if (enable='1') then

AMP_CS <= '1';

SPI_SCK <= '0';

gain_index := 7;

counter := 0;

if(delay=10) then

amp_state <= amp_s2;

delay := 0;

else

delay := delay + 1;

amp_state <= amp_s1;

end if;

else

amp_state <= amp_s1;

end if;

-----amp_state2-----

when amp_s2 =>

if(counter <= 7) then

AMP_CS <= '0';

sPI_MOSI <= gain(gain_index);

if(delay=10) then

amp_state <= amp_s3;

delay := 0;

else

delay := delay + 1;

amp_state <= amp_s2;

end if;

else

AMP_CS <= '1';

state <= adc_states;

end if;

-----amp_state3-----

when amp_s3 =>

SPI_SCK <= '1';

if(delay=10) then

```

amp_state<= amp_s4;
delay :=0;
else
delay := delay+1;
amp_state<=amp_s3;
end if;
-----amp_state 4-----
when amp_s4 =>
SPI_SCK <= '0';
if(delay=10) then
amp_state<= amp_s2;
delay :=0;
-----ADC_state1-----
when adc_s1 =>
If (enable = '0') then
state<= amp_states;
amp_state<= amp_s1;
else
adc_state<= adc_s2;
AD_CONV <= '1';
end if;
-----ADC_state2-----
when adc_s2 =>
AD_CONV <= '0';
counter := 0;
vina_inedx := 16;
adc_state<=adc_s3;
-----ADC_state3-----
when adc_s3 =>
SPI_SCK<='0';
if(delay=2) then
delay :=0;
vina_inedx := vina_inedx -1;
counter := counter+1;
adc_state<=adc_s4;
else
delay := delay+1;
adc_state<=adc_s3;
end if;-----ADC_state4-----
when adc_s4 =>

```

```

gain_index := gain_index-1;
counter:=counter+1;
else
delay := delay+1;
amp_state<=amp_s4;
end if;
-----others amplifier state-----
when others =>
amp_state<= amp_s1;
end case;
whenadc_states =>
caseadc_state is
SPI_SCK <= '1';
if(delay=2) then
delay :=0;
if counter <= 2 then
adc_state<= adc_s3;
elsif counter > 2 and counter <= 16 then
vina(vina_inedx) := SPI_MISO;
adc_state<= adc_s3;
elsif counter <= 33 then
adc_state<= adc_s3;
else
adc_state<= adc_s1;
end if;
else
delay := delay+1;
adc_state<=adc_s4;
end if;
when others =>
adc_state<= adc_s1;
end case;
end case;
end if;
leds<= not vina(13 downto 6);
end process;
end Behavioral

```