XAPP357 (v1.1) January 3, 2002

CoolRunner Visor Springboard LED Test

Summary

LED Test is a simple Springboard™ reference design that allows the Handspring Visor™ to control and blink each of the four LED's on the Insight Springboard™ Development Board. Specifically, four buttons displayed on the Visor are designed to output unique address and data values. The CoolRunner™ CPLD is then used to register (or latch) these values and blink, or turn off, the LEDs accordingly.

This document will detail the C and VHDL code contained in the LED test reference Design. It is intended to help Springboard designers overcome the learning curve associated with the development flow of the CoolRunner XPLA3 CPLD.

All related source code will also be provided for download. To obtain the VHDL code described in this document, go to section **VHDL Code Download**, page 10 for instructions.

LED Test Overview

The block diagram of the LED Test design is shown in Figure 1. For brevity, only the signals that are used in this design are shown. For a complete diagram of the Insight Springboard Development Board, please refer to Insight's documentation or to Xilinx **Application Note XAPP147**.

As shown in Figure 1, the CoolRunner CPLD serves as the central interface between the Handspring Visor and each of the four LED's. The Texas Instruments ADS7870 A/D Converter provides a 2.5 MHz system clock. In order to blink the LEDs the CoolRunner CPLD divides this 2.5 MHz system clock (CCLK) down to a 2.4 Hz signal. This frequency is easily detected by the human eye.

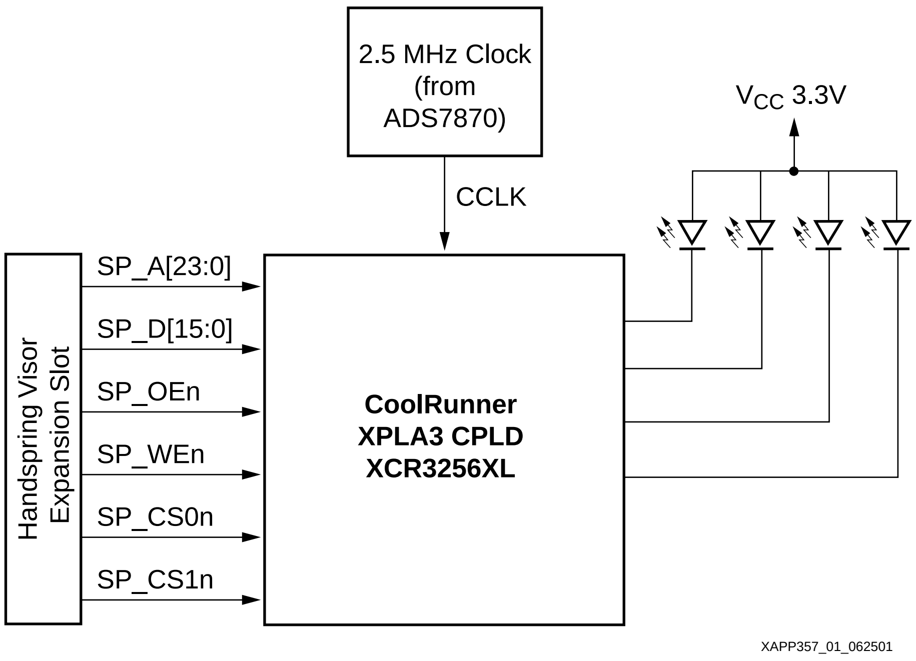
2.5 MHz Clock (from ADS7870)
CCLK
V_{CC} 3.3V
Handspring Visor Expansion Slot
SP_A[23:0]
SP_D[15:0]
SP_OEn
SP_WEn
SP_CS0n
SP_CS1n
CoolRunner XPLA3 CPLD XCR3256XL
XAPP357_01_062501

Figure 1: **LED Test Block Diagram**

C Code

The PocketC development environment is used to create the PalmOS interface because of its ease of use. CControls is used to create the user interface and skeleton code. For documentation on PocketC and CControls, refer to:

www.orbworks.com

The PocketC code shown in Appendix A creates a user interface that consists of four buttons, or switches. Figure 2 shows the user interface.

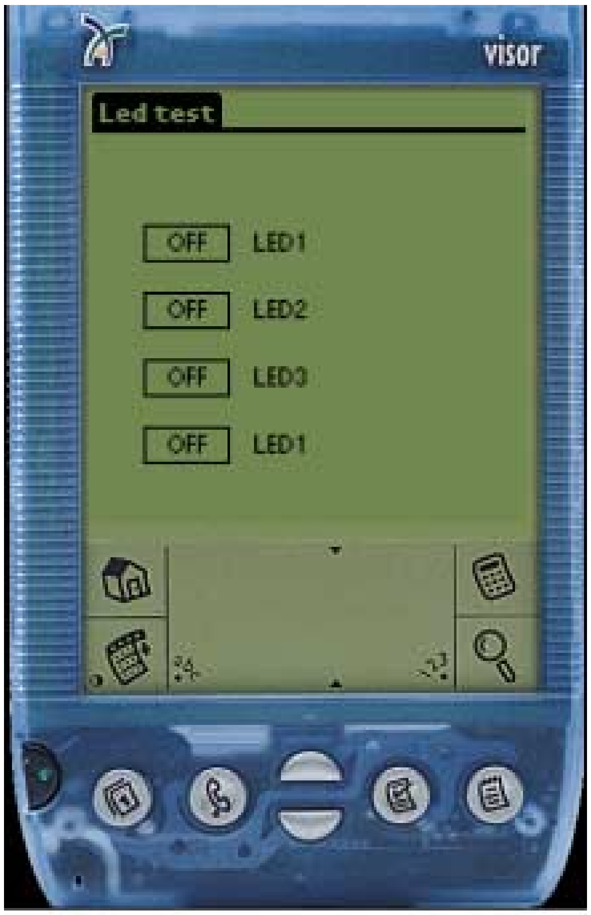

Figure 2: **LED Test User Interface**

Each of the four switches on the Visor's LED screen corresponds to an LED on the Insight Springboard Development Kit. They can be turned either "ON" or "OFF". When turned ON, the Visor outputs a specific address and data value that is qualified by the Write Enable and Chip Select 1 signals.

Note that in order for PocketC to be able to write (or read) to the Springboard, the Xilinx Native Library, "IOLib" must be used. This library contains two basic functions, IOWrite and IORead. IOWrite allows the Visor to write to the Springboard, while IORead allows the visor to read from the Springboard. In this design, IOWrite is used. IOLib is available, free of charge from the Xilinx website.

Table 1 lists the output values with respect to each of the four buttons.

Table 1: **LED Buttons Output Values**

Button	Status	Address	Data
LED1	BLINKING	0x2900003E	0xFFFF
	OFF	0x2900003E	0x0000
LED2	BLINKING	0x29000FC0	0xFFFF
	OFF	0x29000FC0	0x0000
LED3	BLINKING	0x2903F000	0xFFFF
	OFF	0x2903F000	0x0000
LED4	BLINKING	0x29FC0000	0xFFFF
	OFF	0x29FC0000	0x0000

VHDL Code

Given the specifications in Table 1, the VHDL code used by the CoolRunner CPLD must decode the address and data values in order to blink the proper LEDs. The VHDL code shown in Appendix B implements two functions — a simple address/data decoder and a clock divider.

Address/Data Decoder

Four separate processes monitor the Springboard address, data, and control lines. These processes are triggered whenever a write sequence occurs. In other words, they are triggered by the active low signals, SP_WEn (Write Enable) and SP_CS1n (Chip Select 1). Please refer to the Handspring Springboard Developers Guide for a complete timing diagram of a read or a write sequence.

When an ON sequence is detected, one of the four corresponding processes sets a flag for another process to read. When a flag is set, the CoolRunner CPLD blinks an LED with the output of the clock divider. Similarly, when an OFF sequence is detected, a flag is de-asserted and the CPLD turns the corresponding LED off by assigning a logic "1" to that pin.

Clock Divider

In order to blink the LED in a way that can be observed by the eye, the CoolRunner CPLD must divide the 2.5 MHz input clock to a slower rate of 2.4 Hz. This is accomplished by implementing a 20 bit counter clocked by this 2.5 MHz system clock. The MSB of of this counter will be toggling at the desired 2.4 Hz.

Conclusion

This LED Test is a very simple design that illustrates the versatility of the CoolRunner CPLD. Because the Springboard is a pure extension of the Visor's bus, external logic and registers are needed in order to accomplish a simple task like blinking an LED. CoolRunner CPLDs are ideal for this, as they are programmable, abundant in logic resources, and the lowest power CPLD in the market.

Appendix A LED Test C Code

The "LED Test" PocketC source code is shown below. This code creates four buttons that, when used in conjunction with the LED Test VHDL code shown in Appendix B, control each of the four LEDs on the Insight Springboard Development Card. Each button in this application can be turned on or off. For example, looking at the "LED test methods" file, the button named "h1" will output data 0xFFFF to address 0x00003E when turned on. When turned off, the button will output 0x0000 to the same address.

These address values are then decoded by the CoolRunner so that it can blink the appropriate LED.

Note: Three separate files are shown below: "Led test main", "Led test ccontrols", and "Led test methods". These files have been created using CCcontrols.

```
// Led test main (CEditor)
library "IOLib"
include "Led test controls (CEditor)"
include "Led test methods (CEditor)"


main(){
graph_on();
title("Led test");
text(60,50,"LED1");
text(60,75,"LED2");
text(60,100,"LED3");
text(60,125,"LED4");
initcontrols();
initcontents();
inititems();
drawcontrols();
messageloop();
}


/$ Led test controls (CEditor)
include "Ccontrols.c"
Chandle  h1, h2, h3, h4;
initcontrols(){
h1=Cswitch(20,50,30,12,1,0);
h2=Cswitch(20,75,30,12,1,0);
h3=Cswitch(20,100,30,12,1,0);
h4=Cswitch(20,125,30,12,1,0);
}
initcontents(){
Csetcontent(h1,"OFF");
Csetcontent(h2,"OFF");
Csetcontent(h3,"OFF");
Csetcontent(h4,"OFF");
}
inititems(){
}
drawcontrols(){
Cdraw(h1);
Cdraw(h2);
Cdraw(h3);
Cdraw(h4);
}


/$ Led test methods (CEditor)
on_h1(){
```

```
if(Cgetstate(h1)){
     Csetcontent(h1, "ON");
     IOWrite(0x2900003e,0xffff); //write 0xFFFF to address 0x2900003e
      }
else{
     Csetcontent(h1,"OFF");
     IOWrite(0x2900003e,0x0000); //write 0x0000 to address 0x2900003e
     }

Cdraw(h1);

}
on_h2(){
if(Cgetstate(h2)){
     Csetcontent(h2, "ON");
     IOWrite(0x29000fc0,0xffff); //write 0xFFFF to address 0x29000fc0
      }
else{
     Csetcontent(h2,"OFF");
     IOWrite(0x29000fc0,0x0000); //write 0x0000 to address 0x29000fc0
      }
Cdraw(h2);
}

on_h3(){
if(Cgetstate(h3)){
     Csetcontent(h3, "ON");
     IOWrite(0x2903f000,0xffff); //write 0xFFFF to address 0x2903f000

      }
else{
     Csetcontent(h3,"OFF");
     IOWrite(0x2903f000,0x0000); //write 0x0000 to address 0x2903f000

      }
Cdraw(h3);
}

on_h4(){
if(Cgetstate(h4)){
     Csetcontent(h4, "ON");
     IOWrite(0x29fc0000,0xffff); //write 0xFFFF to address 0x29fc0000

      }
else{
     Csetcontent(h4,"OFF");
    IOWrite(0x29fc0000,0x0000); //write 0x0000 to address 0x29fc00000

      }

Cdraw(h4);
}
messageloop(){
int e;
while(1){
e=event(1);
if(Cevent(h1,e)) on_h1();
else if(Cevent(h2,e)) on_h2();
else if(Cevent(h3,e)) on_h3();
else if(Cevent(h4,e)) on_h4();
}
}
```

Appendix B: LED Test VHDL Code

This VHDL code is meant to be used in conjunction with the PocketC code shown in Appendix A. It decodes address and data lines from Handspring Visor and blinks the appropriate LED. The 2.5 MHz CCLK output (generated by Texas Instruments ADS7870 A/D) is divided to generate a 2.4 Hz signal. This 2.4 Hz clock is used to toggle the selected LED.

```
-- *************************************************************
-- File:  led_test.vhd
--
-- Purpose: LED test code for Insight Springboard Development Card.
--          When used in conjunction with the LED test 'C' code, this VHDL
--          code decodes address and data lines from Handspring Visor and
--          Blinks appropriate LED.  The 2.5 MHz CCLK output (generated
--          by Burr-Brown ADS7870 A/D) is divided to generate a 4.76 Hz
--          signal. This 4.76 Hz clock is used to toggle the selected LED
--
-- Date:1-29-2001
--
--
*************************************************************************


library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity LED_TEST is
    port(

          -- Handspring port interface
       SP_A: in STD_LOGIC_VECTOR(23 downto 0); -- 24-bit address bus
       SP_D: in STD_LOGIC_VECTOR(15 downto 0); -- 16-bit data bus
       SP_CS1n: in STD_LOGIC; -- CS1            -- Active low chip select
       SP_WEn: in STD_LOGIC;   -- WE


          -- LED interface
       led1_sel: out  STD_LOGIC; --LED1 pin
       led2_sel: out  STD_LOGIC; --LED2 pin
       led3_sel: out  STD_LOGIC; --LED3 pin
       led4_sel: out  STD_LOGIC; --LED4 pin


            -- ADC interface

    cclk           : in  STD_LOGIC;      -- Output CCLK
    ad_osc_en      : out STD_LOGIC;      -- Oscillator Enable
    ad_convert     : out STD_LOGIC;      -- Convert Pin
    ad_chip2en_n   :out STD_LOGIC;       -- A/D Chip Select
    ad_rise_fall   : out STD_LOGIC;      -- Rise/Fall pin
    serial_tx      : out STD_LOGIC;      -- Din
    serial_clk     : out STD_LOGIC;      -- SClk

          -- Active low write enable

    -- SRAM Interface

   SRAM_CS1n: out STD_LOGIC;           --SRAM CS
    SRAM_UPPER_BYTEn: out STD_LOGIC;   --SRAM Upper Byte Enable
    SRAM_LOW_BYTEn: out STD_LOGIC;     --SRAM Lower Byte Enable

    --Flash Interface
    FLASH_CS0n: out STD_LOGIC;         --Flash CS
    FLASH_WR_PROTECT: out STD_LOGIC;   --Flash /WP/ACC pin

    --SRAM and Flash Interface (shared)

    RWn: out STD_LOGIC;--WE for SRAM and Flash
    OEn: out STD_LOGIC;--OE for SRAM and Flash
    RESETn : out STD_LOGIC;

    CHIP3_EN : out STD_LOGIC--To bring CS out to Expansion Connector

          );

end LED_TEST;



architecture BEHAVIOUR of LED_TEST is

-- ******************** CONSTANT DECLARATIONS ***********************
constant LED1_ADDR : STD_LOGIC_VECTOR(23 downto 0) :=
"000000000000000000111110";  -- A1 to A5 asserted   (2900003E)
constant LED2_ADDR : STD_LOGIC_VECTOR(23 downto 0) :=
"000000000000111111000000";  -- A6 to A11 asserted  (29000FC0)
constant LED3_ADDR : STD_LOGIC_VECTOR(23 downto 0) :=
"000000111111000000000000";  -- A12 to A17 asserted (2903F000)
constant LED4_ADDR : STD_LOGIC_VECTOR(23 downto 0) :=
"111111000000000000000000";  -- A18 to A23 asserted (29FC0000)



-- ********************* SIGNAL DECLARATIONS ***********************
signal toggle_clk  : STD_LOGIC;-- Divided clk for toggling LEDs

signal led1_en : STD_LOGIC;-- LED Enable set/reset by ASSERT_led1_en process
signal led2_en : STD_LOGIC;-- LED Enable set/reset by ASSERT_led2_en process
signal led3_en : STD_LOGIC;-- LED Enable set/reset by ASSERT_led3_en process
signal led4_en : STD_LOGIC;-- LED Enable set/reset by ASSERT_led4_en process



-- ******************** COMPONENT DECLARATION ***********************
-- Clock Divider Function
component CLK_DIVIDER
  port(
    input_clk: in STD_LOGIC;
    output_clk: out STD_LOGIC );

end component;


begin

  -- ***************** SIGNAL ASSIGNMENTS **********************


  -- **************** COMPONENT ASSIGNMENT *********************
  -- Clock Divider Function
  CLK_DIV: CLK_DIVIDER
    port map(
      input_clk=> cclk,
        output_clk=> toggle_clk );



  -- ***************** Process: led1_en ************************
  -- Purpose:  Read address and data lines.  Assert 'led1_en'
  --             when address and data are correct
  -- Components: none

ASSERT_led1_en: process(SP_A, SP_D, SP_WEn, SP_CS1n)
begin
  if(SP_CS1n = '0') then
    if(SP_WEn'event and SP_WEn='0') then
      if (SP_D = "1111111111111111" and SP_A = LED1_ADDR) then
      led1_en <= '1';
      elsif (SP_D = "0000000000000000" and SP_A = LED1_ADDR) then
      led1_en<= '0';
      end if;
    end if;
   end if;
end process ASSERT_led1_en;



  -- ***************** Process: ASSERT_led2_en ************************
  -- Purpose:  Read address and data lines.  Assert 'led2_en'
  --             when address and data are correct
  -- Components: none

ASSERT_led2_en: process(SP_A, SP_D, SP_WEn, SP_CS1n)
begin
  if(SP_CS1n = '0') then
    if(SP_WEn'event and SP_WEn='0') then
      if (SP_D = "1111111111111111" and SP_A = LED2_ADDR) then
     led2_en <= '1';
      elsif (SP_D = "0000000000000000" and SP_A = LED2_ADDR) then
      led2_en<= '0';
      end if;
    end if;
   end if;
end process ASSERT_led2_en;


  -- ***************** Process: ASSERT_led3_en ************************
  -- Purpose:  Read address and data lines.  Assert 'led3_en'
  --             when address and data are correct
  -- Components: none

ASSERT_led3_en: process(SP_A, SP_D, SP_WEn, SP_CS1n)
begin
  if(SP_CS1n = '0') then
    if(SP_WEn'event and SP_WEn='0') then
      if (SP_D = "1111111111111111" and SP_A = LED3_ADDR) then
     led3_en <= '1';
      elsif (SP_D = "0000000000000000" and SP_A = LED3_ADDR) then
      led3_en <= '0';
      end if;
    end if;
   end if;
end process ASSERT_led3_en;
```

```
  -- **************** Process: ASSERT_led4_en ************************
  -- Purpose:  Read address and data lines.  Assert 'led4_en'
  --             when address and data are correct
  -- Components: none

ASSERT_led4_en: process(SP_A, SP_D, SP_WEn, SP_CS1n)
begin
  if(SP_CS1n = '0') then
    if(SP_WEn'event and SP_WEn='0') then
      if (SP_D = "1111111111111111" and SP_A = LED4_ADDR) then
      led4_en <= '1';
      elsif (SP_D = "0000000000000000" and SP_A = LED4_ADDR) then
      led4_en <= '0';
      end if;
    end if;
   end if;
end process ASSERT_led4_en;



led1_sel  <= not (toggle_clk and led1_en);-- Output the LED1 control signal
led2_sel  <= not (toggle_clk and led2_en);-- Output the LED2 control signal
led3_sel  <= not (toggle_clk and led3_en);-- Output the LED3 control signal
led4_sel  <= not (toggle_clk and led4_en);-- Output the LED4 control signal




--SRAM Signals  (Turn OFF the SRAM)
SRAM_UPPER_BYTEn    <= '1';
SRAM_LOW_BYTEn      <= '1';
SRAM_CS1n           <= '1';

--Flash Signals (Turn OFF the Flash)
FLASH_CS0n          <= '1';
FLASH_WR_PROTECT    <= '1';

--Shared Signals (Turn OFF SRAM and Flash)
RWn                 <= '1';
OEn                 <= '1';
RESETn              <= '1';


--A/D Signals    (Turn OFF A/D but enable CCLK output)
ad_osc_en        <= '1';--Osc En pin = '1' to enable CCLK (2.5 MHz)
ad_convert       <= '0';
ad_chip2en_n     <= '1';
ad_rise_fall     <= '0';
serial_tx        <= '0';
serial_clk       <= '0';


--So that CS can be probed
CHIP3_EN <= SP_CS1n;  --Bring CS pin from Springboard Slot to CHIP3_EN pin
           --Not necessary in this demo, but it gives the ability to
           --see what CS is doing.  Also nice to probe with a logic analyzer



end BEHAVIOUR;
```

```
******************************************************************
-- File:  clk_divider.vhd
--
-- Purpose: 20x input clock divider.
**************************************************************************

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;


entity CLK_DIVIDER is
  port(
       input_clk   : in STD_LOGIC;    -- Input Clock  (2.5 MHz Cclk from
                                         ADS7870 A/D)
       output_clk  : out STD_LOGIC); -- Clock divider output


end CLK_DIVIDER;


architecture DEFINITION of CLK_DIVIDER is


-- ******************** SIGNAL DECLARATIONS ***********************
signal div : UNSIGNED (19 downto 0) := (others => '0');

begin

  -- ******************** Process: CNT_PROCESS ************************
  -- Purpose: Defines the 20x counter
  --
  -- Components:none

  CNT_PROCESS:process(input_clk)
  begin

    -- On rising edge of clock
    if (input_clk'event and input_clk = '1') then
      div <= div + 1;
    end if;

      end process;


    output_clk <= div(19);

end DEFINITION;
```

VHDL Code Download

VHDL source code and test benches are available for this design. THE DESIGN IS PROVIDED TO YOU "AS IS". XILINX MAKES AND YOU RECEIVE NO WARRANTIES OR CONDITIONS, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, AND XILINX SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR A PARTICULAR PURPOSE. While this design has been verified on hardware, it should be used only as an example design, not as a fully functional core. XILINX does not warrant the performance, functionality, or operation of this design will meet your requirements, or that the operation of the design will be uninterrupted or error free, or that defects in the design will be corrected. Furthermore, XILINX does not warrant or make any

XAPP357 -http://www.xilinx.com/products/xaw/coolvhdlq.htm

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
09/25/01	1.0	Initial Xilinx release.
01/03/02	1.1	Minor revisions.