About this document

This document will cover the creation and implementation of a simple "ATM Simulator" Trusted System using ARM's TrustZone Security Extensions for the Cortex A9 MPCore Processor Subsystem on the Digilent ZedBoard SoC. The toolchain used will be the Xilinx EDK v14.4 toolchain with CodeSourcery ARM Cross-Compiler that comes on the DVD currently pre-packed with the ZedBoard. While Linux drivers do exist as part of the Linux kernel, in practice these drivers proved problematic or difficult to configure, and a Microsoft Windows 7 development environment was used.

All content within is owned by Sean McClain and Assured Information Security and is (c) 2013 AIS, all rights reserved.

From here on out, the following instruction format will be used:

- → means proceed to the next instruction, expand a checkbox, or hover until another menu appears, where appropriate
- RC Label means to right-click Label and make your selections from the right-click menu
- DC Label means to double-click Label
- TM means top menu
- Tab Label (Location) means an on-screen tab located near Location that matches Label
- Label: contents means to populate the text entry box next to the label with the provided contents
- Label: (un)checked means to (un)check the checkbox next to the label
- Label: selected means to select the radio button choice matching Label
- Label1: Label2: contents means to place the contents at the row Label1, column Label2.
- Label means to click the button with the matching label
- if any values are not specified, you can safely assume the default values are safe to use

For example, in the screen matching figure 1, you might see the following instructions:

- Project name: project_1 → Create Project Subdirectory: checked → Next

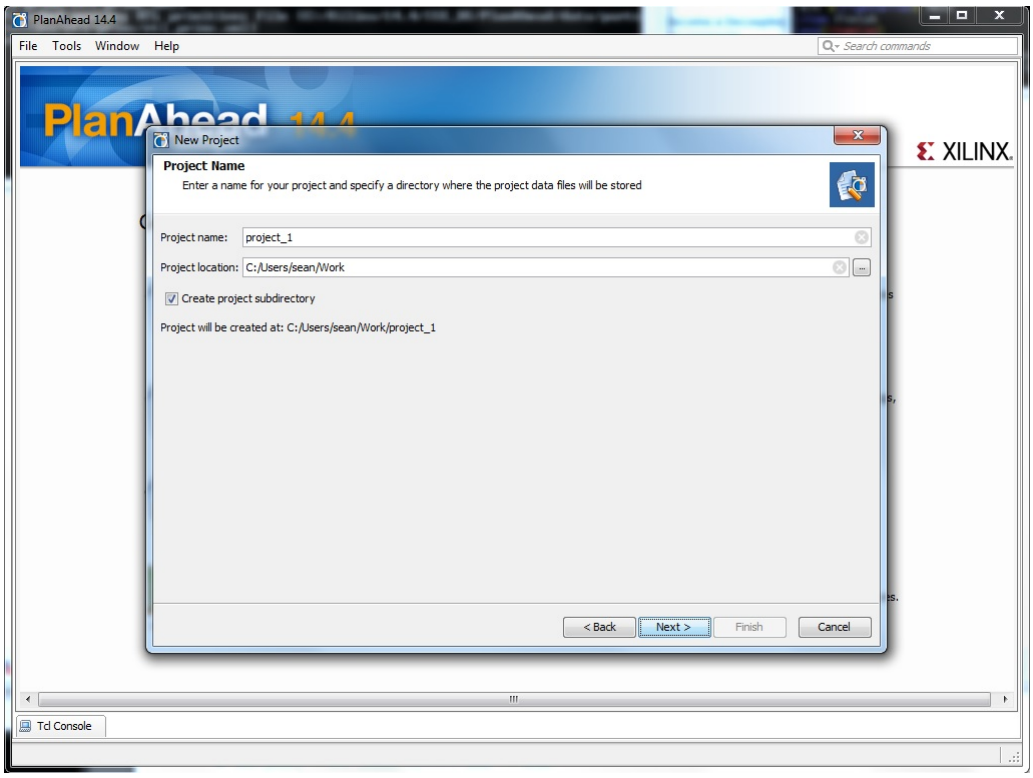

Figure 1: New Project Screen

In this particular example, C:/Users/sean/Work is the "working directory". For the rest of this tutorial, it will be assumed that the working directory is C: . This should be a directory you can locate easily that you have write permissions for.

1 Hardware Design

PlanAhead

To begin, open up the PlanAhead tool (in MS Win7, Xilinx Design Tools → ISE Design Suite 14.4 → PlanAhead → PlanAhead under All Programs in the Start menu).

On the screens that follow, we will create a new PlanAhead project.

- Create New Project
- Next
- Project name: TrustZone → Create project subdirectory: checked → Next
- RTL Project: selected → Do not specify sources at this time: unchecked → Next
- Target language: VHDL → Next
- Next
- Next
- Tab specify (Left) → Boards → Tab Filter (Top) → Package: clg484 → Tab (Bottom) → ZedBoard Zynq Evaluation and Development Kit → Next
- Finish

After a moment, the PlanAhead main window will appear. We will start off by adding a Processor Subsystem (PS). The ZedBoard has two regions: the PS, and the Programmable Logic (PL). The PS comprises everything "built-in" on the board, and the PL consists entirely out of transistor gates you can turn into other electric components.

- Tab Sources (Top Center) → RC Design Sources → Add Sources
- Add or Create Embedded Sources: selected → Next
- Create Sub-Design
- Module name: processor_subsystem → OK
- Finish

XPS

After a moment, Xilinx Platform Studio (XPS) should open. A dialog saying "This project appears to be a blank zynq project. do you want to create a Base System using the BSB Wizard?" will appear. PlanAhead has already been configured, so we can accept all the default options within this wizard. Once the Wizard's completed, we'll add a device in the PL fabric which will act as our "monitor world" for TrustZone. We'll also add in a TrustZone Interrupt Controller.

- If XPS offers to let you use the Base System Builder wizard
  - Yes
  - OK
  - Next
  - Finish
- otherwise
  - If you are prompted to add a processing system, Yes, otherwise, add one manually with Tab IP Catalog (Left) → Processor → DC Processing System (highest IP version) → Yes
  - I/O Peripherals
  - Tab System Assembly View (Bottom) → Tab Zynq (Top) → I/O Peripherals
  - Quad SPI Flash: Enable, MIO 1 .. 6 → Quad SPI Flash : Feedback Clk: Enable, MIO 8
  - Enet 0 : Enable, MIO 16 .. 27 → Enet 0 : MDIO : Enable, MIO 52 .. 53
  - USB 0 : Enable, MIO 28 .. 39
  - SD 0 : Enable, MIO 40 .. 45 → SD 0 : CD : Enable, MIO 47 → SD 0 : WP : Enable, MIO 46
  - UART 1 : Enable, MIO 48 .. 49
  - Timer 0 : Enable, EMIO

  - GPIO : Enable → GPIO : MIO GPIO : Enable, MIO → GPIO : EMIO GPIO (Width) : Enable, 60
  - Close
  - Add Buttons, LEDs, and Switches
  - Tab IP Catalog (Left) → General Purpose IO → DC AXI General Purpose IO → Yes → OK → OK
  - Tab System Assembly View (Bottom) → Tab Bus Interfaces (Top) → single click axi_gpio_0, wait a second, rename to BTNS_5Bits
  - Tab IP Catalog (Left) → General Purpose IO → DC AXI General Purpose IO → Yes → OK → OK
  - Tab System Assembly View (Bottom) → Tab Bus Interfaces (Top) → single click axi_gpio_0, wait a second, rename to LEDs_8Bits
  - Tab IP Catalog (Left) → General Purpose IO → DC AXI General Purpose IO → Yes → OK → OK
  - Tab System Assembly View (Bottom) → Tab Bus Interfaces (Top) → single click axi_gpio_0, wait a second, rename to SWs_8Bits
  - Configure Buttons, LEDs, and Switches
  - Tab System Assembly View (Bottom) → Tab Bus Interfaces (Top) → RC BTNs_5Bits → Configure IP ...
  - Tab User (Right) → Channel 1 → GPIO Data Channel Width: 5 → Channel 1 is Input Only: Checked → OK
  - Tab System Assembly View (Bottom) → Tab Bus Interfaces (Top) → RC LEDs_8Bits → Configure IP ...
  - Tab User (Right) → Channel 1 → GPIO Data Channel Width: 8 → OK
  - Tab System Assembly View (Bottom) → Tab Bus Interfaces (Top) → RC SWs_8Bits → Configure IP ...
  - Tab User (Right) → Channel 1 → GPIO Data Channel Width: 8 → Channel 1 is Input Only: Checked → OK
  - Connect Buttons, LEDs, and Switches
  - Tab System Assembly View (Bottom) → Tab Ports (Top) → BTNs_5Bits → (IO_IF) gpio_0 : Connected Port : Disconnect from External Ports
  - Tab System Assembly View (Bottom) → Tab Ports (Top) → BTNs_5Bits → (IO_IF) gpio_0 : Connected Port : Make Ports External
  - Tab System Assembly View (Bottom) → Tab Ports (Top) → LEDs_8Bits → (IO_IF) gpio_0 : Connected Port : Disconnect from External Ports
  - Tab System Assembly View (Bottom) → Tab Ports (Top) → LEDs_8Bits → (IO_IF) gpio_0 → GPIO_IO_0 : Connected Port : RC → Make External
  - Tab System Assembly View (Bottom) → Tab Ports (Top) → SWs_8Bits → (IO_IF) gpio_0 : Connected Port : Disconnect from External Ports
  - Tab System Assembly View (Bottom) → Tab Ports (Top) → SWs_8Bits → (IO_IF) gpio_0 : Connected Port : Make Ports External
  - Design Rule Check (Ctrl + Shift + D)
  - Tab Console (Bottom) → RC anywhere in the window → Clear Transcript Tabs
  - TM Project → Design Rule Check → Tab Console (Bottom) → Tab Errors (Bottom) → You should have no errors, review these instructions and resolve any that came up.

- Note: If PlanAhead ever informs you that there was an XPS or XST error, but does not specify what the error is, you will want to open XPS, repeat the final two steps in the previous section ("otherwise/Design Rule Check (Ctrl + Shift + D)"), and then TM Hardware → Generate Netlist.
- TM Hardware → Create or Import Peripheral...
- Next
- Create templates for a new peripheral: selected → Next
- Next
- Name: tzac → Description: TrustZone Access Controller → Next
- AXI4-Lite: Selected → Next
- User logic master support: unchecked → Software reset: checked → User logic software register: checked → Include data phase timer: unchecked → Next
- Number of software accessible registers: 3 → Next
- Next

- Generate BFM Simulation Platform: checked → Next
- Generate template driver files to help you implement software interface: checked → Next
- Finish
- Tab IP Catalog (Left) → Project Local PCores → User → DC TZAC
- Yes
- OK
- OK
- Tab System Assembly View (Bottom) → Tab Bus Interfaces (Top) → RC tzac_0 → View MPD
- Replace the contents of tzac_v2_1_0.mpd with the following:

```
##############################################################
##
## Name     : tzac
## Desc     : Microprocessor Peripheral Description
##          : Automatically generated by PsfUtility
##
##############################################################

BEGIN tzac

## Peripheral Options
OPTION IPTYPE = PERIPHERAL
OPTION IMP_NETLIST = TRUE
OPTION HDL = VHDL
OPTION IP_GROUP = MICROBLAZE:USER
OPTION DESC = TZAC
OPTION LONG_DESC = TrustZone Access Controller
OPTION ARCH_SUPPORT_MAP = (others=DEVELOPMENT)


## Bus Interfaces
BUS_INTERFACE BUS = S_AXI, BUS_STD = AXI, BUS_TYPE = SLAVE

## Generics for VHDL or Parameters for Verilog
PARAMETER C_S_AXI_DATA_WIDTH = 32, DT = INTEGER, BUS = S_AXI, ASSIGNMENT = CONSTANT
PARAMETER C_S_AXI_ADDR_WIDTH = 32, DT = INTEGER, BUS = S_AXI, ASSIGNMENT = CONSTANT
PARAMETER C_S_AXI_MIN_SIZE = 0x000001ff, DT = std_logic_vector, BUS = S_AXI
PARAMETER C_USE_WSTRB = 0, DT = INTEGER
PARAMETER C_DPHASE_TIMEOUT = 8, DT = INTEGER
PARAMETER C_BASEADDR = 0xffffffff, DT = std_logic_vector, MIN_SIZE = 0x200, PAIR = C_HIGHADDR, ADDRESS = BASE, BUS = S_AXI
PARAMETER C_HIGHADDR = 0x00000000, DT = std_logic_vector, PAIR = C_BASEADDR, ADDRESS = HIGH, BUS = S_AXI
PARAMETER C_FAMILY = virtex6, DT = STRING
PARAMETER C_NUM_REG = 1, DT = INTEGER
PARAMETER C_NUM_MEM = 1, DT = INTEGER
PARAMETER C_SLV_AWIDTH = 32, DT = INTEGER
PARAMETER C_SLV_DWIDTH = 32, DT = INTEGER
PARAMETER C_S_AXI_PROTOCOL = AXI4LITE, TYPE = NON_HDL, ASSIGNMENT = CONSTANT, DT = STRING, BUS = S_AXI

## Ports
PORT S_AXI_ACLK = "", DIR = I, SIGIS = CLK, BUS = S_AXI
PORT S_AXI_ARESETN = ARESETN, DIR = I, SIGIS = RST, BUS = S_AXI
PORT S_AXI_AWADDR = AWADDR, DIR = I, VEC = [(C_S_AXI_ADDR_WIDTH-1):0], ENDIAN = LITTLE, BUS = S_AXI
PORT S_AXI_AWVALID = AWVALID, DIR = I, BUS = S_AXI
PORT S_AXI_WDATA = WDATA, DIR = I, VEC = [(C_S_AXI_DATA_WIDTH-1):0], ENDIAN = LITTLE, BUS = S_AXI
PORT S_AXI_WSTRB = WSTRB, DIR = I, VEC = [((C_S_AXI_DATA_WIDTH/8)-1):0], ENDIAN = LITTLE, BUS = S_AXI
PORT S_AXI_WVALID = WVALID, DIR = I, BUS = S_AXI
PORT S_AXI_BREADY = BREADY, DIR = I, BUS = S_AXI
PORT S_AXI_ARADDR = ARADDR, DIR = I, VEC = [(C_S_AXI_ADDR_WIDTH-1):0], ENDIAN = LITTLE, BUS = S_AXI
PORT S_AXI_ARVALID = ARVALID, DIR = I, BUS = S_AXI
PORT S_AXI_RREADY = RREADY, DIR = I, BUS = S_AXI
PORT S_AXI_ARREADY = ARREADY, DIR = O, BUS = S_AXI
PORT S_AXI_RDATA = RDATA, DIR = O, VEC = [(C_S_AXI_DATA_WIDTH-1):0], ENDIAN = LITTLE, BUS = S_AXI
PORT S_AXI_RRESP = RRESP, DIR = O, VEC = [1:0], BUS = S_AXI
PORT S_AXI_RVALID = RVALID, DIR = O, BUS = S_AXI
PORT S_AXI_WREADY = WREADY, DIR = O, BUS = S_AXI
PORT S_AXI_BRESP = BRESP, DIR = O, VEC = [1:0], BUS = S_AXI
PORT S_AXI_BVALID = BVALID, DIR = O, BUS = S_AXI
PORT S_AXI_AWREADY = AWREADY, DIR = O, BUS = S_AXI
PORT S_AXI_AWPROT = AWPROT, DIR = I, VEC = [2:0], BUS = S_AXI
PORT S_AXI_ARPROT = AWPROT, DIR = I, VEC = [2:0], BUS = S_AXI
PORT IRQ = "", SIGIS = INTERRUPT, DIR = O, SENSITIVITY = LEVEL_HIGH

END
```

Explanation: We added the 3 lines at the bottom. AWPROT and AXPROT are the TrustZone AXI signals. And the last line will allow our device to trigger interrupts.

- Tab System Assembly View (Bottom) → Tab Bus Interfaces (Top) → RC tzac_0 → Browse HDL Sources
- Hold down shift and select both files → Open
- Replace the contents of user_logic.vhd with the following, to ensure registers can only be written to from secure world, and only register 3 can be read from normal world:

```
------------------------------------------------------------------------------
-- user_logic.vhd - entity/architecture pair
------------------------------------------------------------------------------
--
-- ***************************************************************************
-- ** Copyright (c) 1995-2012 Xilinx, Inc.  All rights reserved.            **
-- **                                                                       **
-- ** Xilinx, Inc.                                                          **
-- ** XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS"         **
-- ** AS A COURTESY TO YOU, SOLELY FOR USE IN DEVELOPING PROGRAMS AND       **
-- ** SOLUTIONS FOR XILINX DEVICES.  BY PROVIDING THIS DESIGN, CODE,        **
-- ** OR INFORMATION AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE,        **
-- ** APPLICATION OR STANDARD, XILINX IS MAKING NO REPRESENTATION           **
-- ** THAT THIS IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT,     **
-- ** AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE      **
-- ** FOR YOUR IMPLEMENTATION.  XILINX EXPRESSLY DISCLAIMS ANY              **
-- ** WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE               **
```

```
-- ** IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR        **
-- ** REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF       **
-- ** INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS       **
-- ** FOR A PARTICULAR PURPOSE.                                             **
-- **                                                                      **
-- **************************************************************************
--
----------------------------------------------------------------------------
-- Filename:          user_logic.vhd
-- Version:           1.00.a
-- Description:       User logic.
-- Date:              Mon Sep 23 16:22:31 2013 (by Create and Import Peripheral Wizard)
-- VHDL Standard:     VHDL'93
--
-- The TrustZone Access Controller peripheral has 3 registers. They are designed so that there is at least 1 bit which can port functionality to or from each of the TrustZone registers. Normal world may
--
--     R0[31:16] : Normal World Data 1
--     R0[15:0] : Normal World Data 2
--     R1[31:21] : Not used
--     R1[20:14] : Correspond to bits in NSACR
--         R1[ 20 ] : NS_SMP
--         R1[ 19 ] : TL
--         R1[ 18 ] : PLE
--         R1[ 17 ] : NSASEDIS
--         R1[ 16 ] : NSD32DIS
--         R1[ 15 ] : CP11
--         R1[ 14 ] : CP10
--     R1[13:12] : Correspond to bits in SDER
--         R1[ 13 ] : SUNIDEN
--         R1[ 12 ] : SUIDEN
--     R1[11:10] : Correspond to AxPROT(1) bits coming into the peripheral
--         R[ 11 ] : ARPROT
--         R[ 10 ] : AWPROT
--     R1[9:0] : Correspond to bits in SCR
--         R[ 9 ] : SIF
--         R[ 8 ] : HCE
--         R[ 7 ] : SCD
--         R[ 6 ] : nET
--         R[ 5 ] : AW
--         R[ 4 ] : FW
--         R[ 3 ] : EA
--         R[ 2 ] : FIQ
--         R[ 1 ] : IRQ
--         R[ 0 ] : NS
--     R2[31:16] : Secure World Data
--     R2[15:12] : SAC bits [3:0]
--     R2[11:0] correspond directly to bits in the SNSAC
--
----------------------------------------------------------------------------
-- Naming Conventions:
--   active low signals:                    "*_n"
--   clock signals:                         "clk", "clk_div#", "clk_#x"
--   reset signals:                         "rst", "rst_n"
--   generics:                              "C_*"
--   user defined types:                    "*_TYPE"
--   state machine next state:              "*_ns"
--   state machine current state:           "*_cs"
--   combinatorial signals:                 "*_com"
--   pipelined or register delay signals:   "*_d#"
--   counter signals:                       "*cnt*"
--   clock enable signals:                  "*_ce"
--   internal version of output port:       "*_i"
--   device pins:                           "*_pin"
--   ports:                                 "- Names begin with Uppercase"
--   processes:                             "*_PROCESS"
--   component instantiations:              "<ENTITY_>I_<#|FUNC>"
----------------------------------------------------------------------------

-- DO NOT EDIT BELOW THIS LINE --------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

library proc_common_v3_00_a;
use proc_common_v3_00_a.proc_common_pkg.all;

-- DO NOT EDIT ABOVE THIS LINE --------------------

--USER libraries added here

----------------------------------------------------------------------------
-- Entity section
----------------------------------------------------------------------------
-- Definition of Generics:
--   C_NUM_REG                    -- Number of software accessible registers
--   C_SLV_DWIDTH                 -- Slave interface data bus width
--
-- Definition of Ports:
--   Bus2IP_Clk                   -- Bus to IP clock
--   Bus2IP_Resetn                -- Bus to IP reset
--   Bus2IP_Data                  -- Bus to IP data bus
--   Bus2IP_BE                    -- Bus to IP byte enables
--   Bus2IP_RdCE                  -- Bus to IP read chip enable
--   Bus2IP_WrCE                  -- Bus to IP write chip enable
--   IP2Bus_Data                  -- IP to Bus data bus
--   IP2Bus_RdAck                 -- IP to Bus read transfer acknowledgement
--   IP2Bus_WrAck                 -- IP to Bus write transfer acknowledgement
--   IP2Bus_Error                 -- IP to Bus error response
----------------------------------------------------------------------------

entity user_logic is
  generic
  (
    -- ADD USER GENERICS BELOW THIS LINE ---------------
    --USER generics added here
    -- ADD USER GENERICS ABOVE THIS LINE ---------------

    -- DO NOT EDIT BELOW THIS LINE ---------------------
    -- Bus protocol parameters, do not add to or delete
    C_NUM_REG                      : integer              := 3;
    C_SLV_DWIDTH                   : integer              := 32
    -- DO NOT EDIT ABOVE THIS LINE ---------------------
  );
  port
  (
    -- ADD USER PORTS BELOW THIS LINE ------------------
    S_AXI_AWPROT                   : in  std_logic_vector(2 downto 0);
    S_AXI_ARPROT                   : in  std_logic_vector(2 downto 0);
    IRQ                            : out std_logic;
    -- ADD USER PORTS ABOVE THIS LINE ------------------

    -- DO NOT EDIT BELOW THIS LINE ---------------------
    -- Bus protocol ports, do not add to or delete
    Bus2IP_Clk                     : in  std_logic;
    Bus2IP_Resetn                  : in  std_logic;
    Bus2IP_Data                    : in  std_logic_vector(C_SLV_DWIDTH-1 downto 0);
    Bus2IP_BE                      : in  std_logic_vector(C_SLV_DWIDTH/8-1 downto 0);
```

```
    Bus2IP_RdCE                    : in  std_logic_vector(C_NUM_REG-1 downto 0);
    Bus2IP_WrCE                    : in  std_logic_vector(C_NUM_REG-1 downto 0);
    IP2Bus_Data                    : out std_logic_vector(C_SLV_DWIDTH-1 downto 0);
    IP2Bus_RdAck                   : out std_logic;
    IP2Bus_WrAck                   : out std_logic;
    IP2Bus_Error                   : out std_logic
    -- DO NOT EDIT ABOVE THIS LINE ---------------------
  );

  attribute MAX_FANOUT : string;
  attribute SIGIS : string;

  attribute SIGIS of Bus2IP_Clk    : signal is "CLK";
  attribute SIGIS of Bus2IP_Resetn : signal is "RST";

end entity user_logic;

------------------------------------------------------------------------------
-- Architecture section
------------------------------------------------------------------------------

architecture IMP of user_logic is

  --USER signal declarations added here, as needed for user logic

  ---------------------------------------------
  -- Signals for user logic slave model s/w accessible register example
  ---------------------------------------------
  signal slv_reg0                        : std_logic_vector(C_SLV_DWIDTH-1 downto 0);
  signal slv_reg1                        : std_logic_vector(C_SLV_DWIDTH-1 downto 0);
  signal slv_reg2                        : std_logic_vector(C_SLV_DWIDTH-1 downto 0);
  signal slv_reg_write_sel               : std_logic_vector(2 downto 0);
  signal slv_reg_read_sel                : std_logic_vector(2 downto 0);
  signal slv_ip2bus_data                 : std_logic_vector(C_SLV_DWIDTH-1 downto 0);
  signal slv_read_ack                    : std_logic;
  signal slv_write_ack                   : std_logic;

begin

  --USER logic implementation added here

  ---------------------------------------------
  -- Example code to read/write user logic slave model s/w accessible registers
  --
  -- Note:
  -- The example code presented here is to show you one way of reading/writing
  -- software accessible registers implemented in the user logic slave model.
  -- Each bit of the Bus2IP_WrCE/Bus2IP_RdCE signals is configured to correspond
  -- to one software accessible register by the top level template. For example,
  -- if you have four 32 bit software accessible registers in the user logic,
  -- you are basically operating on the following memory mapped registers:
  --
  --    Bus2IP_WrCE/Bus2IP_RdCE   Memory Mapped Register
  --                     "1000"   C_BASEADDR + 0x0
  --                     "0100"   C_BASEADDR + 0x4
  --                     "0010"   C_BASEADDR + 0x8
  --                     "0001"   C_BASEADDR + 0xC
  --
  ---------------------------------------------
  slv_reg_write_sel <= Bus2IP_WrCE(2 downto 0);
  slv_reg_read_sel  <= Bus2IP_RdCE(2 downto 0);
  slv_write_ack     <= Bus2IP_WrCE(0) or Bus2IP_WrCE(1) or Bus2IP_WrCE(2);
  slv_read_ack      <= Bus2IP_RdCE(0) or Bus2IP_RdCE(1) or Bus2IP_RdCE(2);

  -- implement slave model software accessible register(s)
  SLAVE_REG_WRITE_PROC : process( Bus2IP_Clk ) is
    variable slv_reg_write_sel_secure : std_logic_vector(C_NUM_REG downto 0);
  begin

    slv_reg_write_sel_secure(C_NUM_REG) := S_AXI_AWPROT(1);
    slv_reg_write_sel_secure(C_NUM_REG - 1 downto 0) := slv_reg_write_sel;

    check_high_edge: if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
      -- reset
      check_reset: if Bus2IP_Resetn = '0' then
        slv_reg0 <= (others => '0');
        slv_reg1 <= (others => '0');
        slv_reg2 <= (others => '0');

      -- IRQ clear
      elsif Bus2IP_Data(1) = '0' and slv_reg_write_sel_secure = "1010" then
        slv_reg1(1) <= '0';
      -- uncomment for non-secure access
      elsif Bus2IP_Data(1) = '0' and slv_reg_write_sel_secure = "0010" then
        slv_reg1(1) <= '0';

      -- potential read or write
      else

        -- R0: Normal World Data
        check_write_reg: if slv_reg_write_sel_secure = "1100" then
          slv_reg0(31 downto 16) <= Bus2IP_Data(31 downto 16); -- normal world data 1
          slv_reg0(15 downto 0) <= Bus2IP_Data(15 downto 0); -- normal world data 2
          slv_reg1(1) <= '1'; -- flag IRQ any time there is a write
          slv_reg1(10) <= S_AXI_AWPROT(1); -- mark whether this write was secure
        elsif slv_reg_write_sel_secure = "0100" then
          slv_reg0(31 downto 16) <= Bus2IP_Data(31 downto 16); -- normal world data 1
          slv_reg0(15 downto 0) <= Bus2IP_Data(15 downto 0); -- normal world data 2
          slv_reg1(1) <= '1'; -- flag IRQ any time there is a write
          slv_reg1(10) <= S_AXI_AWPROT(1); -- mark whether this write was secure

        -- uncomment to allow unrestricted access to R1 and R2
        elsif slv_reg_write_sel_secure = "0010" then
          slv_reg1(0) <= Bus2IP_Data(0);
          slv_reg1(1) <= '1'; -- flag IRQ any time there is a write
 slv_reg1(9 downto 2) <= Bus2IP_Data(9 downto 2);
          slv_reg1(10) <= S_AXI_AWPROT(1); -- mark whether this write was secure
 slv_reg1(31 downto 11) <= Bus2IP_Data(31 downto 11);
        elsif slv_reg_write_sel_secure = "0001" then
          slv_reg2 <= Bus2IP_Data;
          slv_reg1(1) <= '1'; -- flag IRQ any time there is a write
          slv_reg1(10) <= S_AXI_AWPROT(1); -- mark whether this write was secure

        -- R1: TrustZone config bits
        elsif slv_reg_write_sel_secure = "1010" then
          -- SCR
          slv_reg1(0) <= Bus2IP_Data(0); -- NS
          slv_reg1(2) <= Bus2IP_Data(2); -- FIQ
          slv_reg1(3) <= Bus2IP_Data(3); -- EA
          slv_reg1(4) <= Bus2IP_Data(4); -- FW
          slv_reg1(5) <= Bus2IP_Data(5); -- AW
          slv_reg1(6) <= Bus2IP_Data(6); -- nET
          slv_reg1(7) <= Bus2IP_Data(7); -- SCD
          slv_reg1(8) <= Bus2IP_Data(8); -- HCE
          slv_reg1(9) <= Bus2IP_Data(9); -- SIF

          -- SDER
          slv_reg1(12) <= Bus2IP_Data(12); -- SUIDEN
```

```
          slv_reg1(13) <= Bus2IP_Data(13); -- SUNIDEN

          -- NSACR
          slv_reg1(14) <= Bus2IP_Data(14); -- CP10
          slv_reg1(15) <= Bus2IP_Data(15); -- CP11
          slv_reg1(16) <= Bus2IP_Data(16); -- NSD32DIS
          slv_reg1(17) <= Bus2IP_Data(17); -- NSASEDIS
          slv_reg1(18) <= Bus2IP_Data(18); -- PLE
          slv_reg1(19) <= Bus2IP_Data(19); -- TL
          slv_reg1(20) <= Bus2IP_Data(20); -- NS_SMP

          -- Not used
          slv_reg1(31 downto 21) <= (others => '0');

          slv_reg1(1) <= '1'; -- flag IRQ any time there is a write
          slv_reg1(10) <= S_AXI_AWPROT(1); -- mark whether this write was secure

        -- R2: SCU config bits and Secure Data
        elsif slv_reg_write_sel_secure = "1001" then
          slv_reg2(31 downto 16) <= Bus2IP_Data(31 downto 16); -- Secure Data
          slv_reg2(15 downto 12) <= Bus2IP_Data(15 downto 12); -- SAC[3:0]
          slv_reg2(11 downto 0) <= Bus2IP_Data(11 downto 0); -- SNSAC[11:0]

          slv_reg1(1) <= '1'; -- flag IRQ any time there is a write
          slv_reg1(10) <= S_AXI_AWPROT(1); -- mark whether this write was secure

        -- mark whether there is a secure read or not
        elsif slv_reg_read_sel = "100" then
          slv_reg1(11) <= S_AXI_ARPROT(1);
        elsif slv_reg_read_sel = "010" then
          slv_reg1(11) <= S_AXI_ARPROT(1);
        elsif slv_reg_read_sel = "001" then
          slv_reg1(11) <= S_AXI_ARPROT(1);

        end if check_write_reg;

      end if check_reset;

-- On every cycle
      IRQ <= slv_reg1(1); -- Copy out the stored IRQ value

    end if check_high_edge;

  end process SLAVE_REG_WRITE_PROC;

  -- implement slave model software accessible register(s) read mux
  SLAVE_REG_READ_PROC : process( slv_reg_read_sel, slv_reg0, slv_reg1, slv_reg2 ) is
    variable slv_reg_read_sel_secure : std_logic_vector(C_NUM_REG downto 0);
  begin

    slv_reg_read_sel_secure(C_NUM_REG) := S_AXI_ARPROT(1);
    slv_reg_read_sel_secure(C_NUM_REG - 1 downto 0) := slv_reg_read_sel;

    case slv_reg_read_sel_secure is
      when "0100" => slv_ip2bus_data <= slv_reg0;
      when "0010" => slv_ip2bus_data <= slv_reg1;
      when "0001" => slv_ip2bus_data <= slv_reg2;
      when "1100" => slv_ip2bus_data <= slv_reg0;
      when "1010" => slv_ip2bus_data <= slv_reg1;
      when "1001" => slv_ip2bus_data <= slv_reg2;
      when others => slv_ip2bus_data <= (others => '0');
    end case;

  end process SLAVE_REG_READ_PROC;

  ------------------------------------------
  -- Example code to drive IP to Bus signals
  ------------------------------------------
  IP2Bus_Data  <= slv_ip2bus_data when slv_read_ack = '1' else
                  (others => '0');

  IP2Bus_WrAck <= slv_write_ack;
  IP2Bus_RdAck <= slv_read_ack;
  IP2Bus_Error <= '0';

end IMP;
```

- Next to user_logic.vhd, replace the contents of tzac.vhd with the following to enable TrustZone Security Extensions and our IRQ signal:

```
------------------------------------------------------------------------------
-- tzac.vhd - entity/architecture pair
------------------------------------------------------------------------------
-- IMPORTANT:
-- DO NOT MODIFY THIS FILE EXCEPT IN THE DESIGNATED SECTIONS.
--
-- SEARCH FOR --USER TO DETERMINE WHERE CHANGES ARE ALLOWED.
--
-- TYPICALLY, THE ONLY ACCEPTABLE CHANGES INVOLVE ADDING NEW
-- PORTS AND GENERICS THAT GET PASSED THROUGH TO THE INSTANTIATION
-- OF THE USER_LOGIC ENTITY.
------------------------------------------------------------------------------
--
-- ***************************************************************************
-- ** Copyright (c) 1995-2012 Xilinx, Inc.  All rights reserved.            **
-- **                                                                       **
-- ** Xilinx, Inc.                                                          **
-- ** XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS"         **
-- ** AS A COURTESY TO YOU, SOLELY FOR USE IN DEVELOPING PROGRAMS AND       **
-- ** SOLUTIONS FOR XILINX DEVICES.  BY PROVIDING THIS DESIGN, CODE,        **
-- ** OR INFORMATION AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE,        **
-- ** APPLICATION OR STANDARD, XILINX IS MAKING NO REPRESENTATION           **
-- ** THAT THIS IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT,     **
-- ** AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE      **
-- ** FOR YOUR IMPLEMENTATION.  XILINX EXPRESSLY DISCLAIMS ANY              **
-- ** WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE               **
-- ** IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR        **
-- ** REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF       **
-- ** INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS       **
-- ** FOR A PARTICULAR PURPOSE.                                             **
-- **                                                                       **
-- ***************************************************************************
--
------------------------------------------------------------------------------
-- Filename:          tzac.vhd
-- Version:           1.00.a
-- Description:       Top level design, instantiates library components and user logic.
-- Date:              Mon Sep 23 16:22:31 2013 (by Create and Import Peripheral Wizard)
-- VHDL Standard:     VHDL'93
------------------------------------------------------------------------------
-- Naming Conventions:
--   active low signals:                    "*_n"
--   clock signals:                         "clk", "clk_div#", "clk_#x"
--   reset signals:                         "rst", "rst_n"
--   generics:                              "C_*"
```

```
--   user defined types:                    "*_TYPE"
--   state machine next state:              "*_ns"
--   state machine current state:           "*_cs"
--   combinatorial signals:                 "*_com"
--   pipelined or register delay signals:   "*_d#"
--   counter signals:                       "*cnt*"
--   clock enable signals:                  "*_ce"
--   internal version of output port:       "*_i"
--   device pins:                           "*_pin"
--   ports:                                 "- Names begin with Uppercase"
--   processes:                             "*_PROCESS"
--   component instantiations:              "<ENTITY_>I_<#|FUNC>"
------------------------------------------------------------------------------

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

library proc_common_v3_00_a;
use proc_common_v3_00_a.proc_common_pkg.all;
use proc_common_v3_00_a.ipif_pkg.all;
use proc_common_v3_00_a.soft_reset;

library axi_lite_ipif_v1_01_a;
use axi_lite_ipif_v1_01_a.axi_lite_ipif;

library tzac_v1_00_a;
use tzac_v1_00_a.user_logic;

------------------------------------------------------------------------------
-- Entity section
------------------------------------------------------------------------------
-- Definition of Generics:
--   C_S_AXI_DATA_WIDTH           -- AXI4LITE slave: Data width
--   C_S_AXI_ADDR_WIDTH           -- AXI4LITE slave: Address Width
--   C_S_AXI_MIN_SIZE             -- AXI4LITE slave: Min Size
--   C_USE_WSTRB                  -- AXI4LITE slave: Write Strobe
--   C_DPHASE_TIMEOUT             -- AXI4LITE slave: Data Phase Timeout
--   C_BASEADDR                   -- AXI4LITE slave: base address
--   C_HIGHADDR                   -- AXI4LITE slave: high address
--   C_FAMILY                     -- FPGA Family
--   C_NUM_REG                    -- Number of software accessible registers
--   C_NUM_MEM                    -- Number of address-ranges
--   C_SLV_AWIDTH                 -- Slave interface address bus width
--   C_SLV_DWIDTH                 -- Slave interface data bus width
--
-- Definition of Ports:
--   S_AXI_ACLK                   -- AXI4LITE slave: Clock
--   S_AXI_ARESETN                -- AXI4LITE slave: Reset
--   S_AXI_AWADDR                 -- AXI4LITE slave: Write address
--   S_AXI_AWVALID                -- AXI4LITE slave: Write address valid
--   S_AXI_WDATA                  -- AXI4LITE slave: Write data
--   S_AXI_WSTRB                  -- AXI4LITE slave: Write strobe
--   S_AXI_WVALID                 -- AXI4LITE slave: Write data valid
--   S_AXI_BREADY                 -- AXI4LITE slave: Response ready
--   S_AXI_ARADDR                 -- AXI4LITE slave: Read address
--   S_AXI_ARVALID                -- AXI4LITE slave: Read address valid
--   S_AXI_RREADY                 -- AXI4LITE slave: Read data ready
--   S_AXI_ARREADY                -- AXI4LITE slave: read addres ready
--   S_AXI_RDATA                  -- AXI4LITE slave: Read data
--   S_AXI_RRESP                  -- AXI4LITE slave: Read data response
--   S_AXI_RVALID                 -- AXI4LITE slave: Read data valid
--   S_AXI_WREADY                 -- AXI4LITE slave: Write data ready
--   S_AXI_BRESP                  -- AXI4LITE slave: Response
--   S_AXI_BVALID                 -- AXI4LITE slave: Resonse valid
--   S_AXI_AWREADY                -- AXI4LITE slave: Wrte address ready
------------------------------------------------------------------------------

entity tzac is
  generic
  (
    -- ADD USER GENERICS BELOW THIS LINE ---------------
    --USER generics added here
    -- ADD USER GENERICS ABOVE THIS LINE ---------------

    -- DO NOT EDIT BELOW THIS LINE ---------------------
    -- Bus protocol parameters, do not add to or delete
    C_S_AXI_DATA_WIDTH             : integer              := 32;
    C_S_AXI_ADDR_WIDTH             : integer              := 32;
    C_S_AXI_MIN_SIZE               : std_logic_vector     := X"000001FF";
    C_USE_WSTRB                    : integer              := 0;
    C_DPHASE_TIMEOUT               : integer              := 8;
    C_BASEADDR                     : std_logic_vector     := X"FFFFFFFF";
    C_HIGHADDR                     : std_logic_vector     := X"00000000";
    C_FAMILY                       : string               := "virtex6";
    C_NUM_REG                      : integer              := 1;
    C_NUM_MEM                      : integer              := 1;
    C_SLV_AWIDTH                   : integer              := 32;
    C_SLV_DWIDTH                   : integer              := 32
    -- DO NOT EDIT ABOVE THIS LINE ---------------------
  );
  port
  (
    -- ADD USER PORTS BELOW THIS LINE ------------------
    S_AXI_AWPROT                   : in  std_logic_vector(2 downto 0);
    S_AXI_ARPROT                   : in  std_logic_vector(2 downto 0);
    IRQ                            : out std_logic;
    -- ADD USER PORTS ABOVE THIS LINE ------------------

    -- DO NOT EDIT BELOW THIS LINE ---------------------
    -- Bus protocol ports, do not add to or delete
    S_AXI_ACLK                     : in  std_logic;
    S_AXI_ARESETN                  : in  std_logic;
    S_AXI_AWADDR                   : in  std_logic_vector(C_S_AXI_ADDR_WIDTH-1 downto 0);
    S_AXI_AWVALID                  : in  std_logic;
    S_AXI_WDATA                    : in  std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
    S_AXI_WSTRB                    : in  std_logic_vector((C_S_AXI_DATA_WIDTH/8)-1 downto 0);
    S_AXI_WVALID                   : in  std_logic;
    S_AXI_BREADY                   : in  std_logic;
    S_AXI_ARADDR                   : in  std_logic_vector(C_S_AXI_ADDR_WIDTH-1 downto 0);
    S_AXI_ARVALID                  : in  std_logic;
    S_AXI_RREADY                   : in  std_logic;
    S_AXI_ARREADY                  : out std_logic;
    S_AXI_RDATA                    : out std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
    S_AXI_RRESP                    : out std_logic_vector(1 downto 0);
    S_AXI_RVALID                   : out std_logic;
    S_AXI_WREADY                   : out std_logic;
    S_AXI_BRESP                    : out std_logic_vector(1 downto 0);
    S_AXI_BVALID                   : out std_logic;
    S_AXI_AWREADY                  : out std_logic
    -- DO NOT EDIT ABOVE THIS LINE ---------------------
  );

  attribute MAX_FANOUT : string;
  attribute SIGIS : string;
  attribute MAX_FANOUT of S_AXI_ACLK       : signal is "10000";
  attribute MAX_FANOUT of S_AXI_ARESETN       : signal is "10000";
```

```
  attribute SIGIS of S_AXI_ACLK       : signal is "Clk";
  attribute SIGIS of S_AXI_ARESETN       : signal is "Rst";
end entity tzac;

------------------------------------------------------------------------------
-- Architecture section
------------------------------------------------------------------------------

architecture IMP of tzac is

  constant USER_SLV_DWIDTH                : integer              := C_S_AXI_DATA_WIDTH;

  constant IPIF_SLV_DWIDTH                : integer              := C_S_AXI_DATA_WIDTH;

  constant ZERO_ADDR_PAD                  : std_logic_vector(0 to 31) := (others => '0');
  constant RST_BASEADDR                   : std_logic_vector     := C_BASEADDR or X"00000100";
  constant RST_HIGHADDR                   : std_logic_vector     := C_BASEADDR or X"000001FF";
  constant USER_SLV_BASEADDR              : std_logic_vector     := C_BASEADDR or X"00000000";
  constant USER_SLV_HIGHADDR              : std_logic_vector     := C_BASEADDR or X"000000FF";

  constant IPIF_ARD_ADDR_RANGE_ARRAY      : SLV64_ARRAY_TYPE     :=
    (
      ZERO_ADDR_PAD & RST_BASEADDR,       -- soft reset space base address
      ZERO_ADDR_PAD & RST_HIGHADDR,       -- soft reset space high address
      ZERO_ADDR_PAD & USER_SLV_BASEADDR,  -- user logic slave space base address
      ZERO_ADDR_PAD & USER_SLV_HIGHADDR   -- user logic slave space high address
    );

  constant RST_NUM_CE                     : integer              := 1;
  constant USER_SLV_NUM_REG               : integer              := 3;
  constant USER_NUM_REG                   : integer              := USER_SLV_NUM_REG;
  constant TOTAL_IPIF_CE                  : integer              := USER_NUM_REG + RST_NUM_CE;

  constant IPIF_ARD_NUM_CE_ARRAY          : INTEGER_ARRAY_TYPE   :=
    (
      0  => (RST_NUM_CE),                 -- number of ce for soft reset space
      1  => (USER_SLV_NUM_REG)            -- number of ce for user logic slave space
    );

  ------------------------------------------
  -- Width of triggered reset in bus clocks
  ------------------------------------------
  constant RESET_WIDTH                    : integer              := 8;

  ------------------------------------------
  -- Index for CS/CE
  ------------------------------------------
  constant RST_CS_INDEX                   : integer              := 0;
  constant RST_CE_INDEX                   : integer              := USER_NUM_REG;
  constant USER_SLV_CS_INDEX              : integer              := 1;
  constant USER_SLV_CE_INDEX              : integer              := calc_start_ce_index(IPIF_ARD_NUM_CE_ARRAY, USER_SLV_CS_INDEX);

  constant USER_CE_INDEX                  : integer              := USER_SLV_CE_INDEX;

  ------------------------------------------
  -- IP Interconnect (IPIC) signal declarations
  ------------------------------------------
  signal ipif_Bus2IP_Clk                : std_logic;
  signal ipif_Bus2IP_Resetn             : std_logic;
  signal ipif_Bus2IP_Addr               : std_logic_vector(C_S_AXI_ADDR_WIDTH-1 downto 0);
  signal ipif_Bus2IP_RNW                : std_logic;
  signal ipif_Bus2IP_BE                 : std_logic_vector(IPIF_SLV_DWIDTH/8-1 downto 0);
  signal ipif_Bus2IP_CS                 : std_logic_vector((IPIF_ARD_ADDR_RANGE_ARRAY'LENGTH)/2-1 downto 0);
  signal ipif_Bus2IP_RdCE               : std_logic_vector(calc_num_ce(IPIF_ARD_NUM_CE_ARRAY)-1 downto 0);
  signal ipif_Bus2IP_WrCE               : std_logic_vector(calc_num_ce(IPIF_ARD_NUM_CE_ARRAY)-1 downto 0);
  signal ipif_Bus2IP_Data               : std_logic_vector(IPIF_SLV_DWIDTH-1 downto 0);
  signal ipif_IP2Bus_WrAck              : std_logic;
  signal ipif_IP2Bus_RdAck              : std_logic;
  signal ipif_IP2Bus_Error              : std_logic;
  signal ipif_IP2Bus_Data               : std_logic_vector(IPIF_SLV_DWIDTH-1 downto 0);
  signal ipif_Bus2IP_Reset              : std_logic;
  signal rst_Bus2IP_Reset               : std_logic;
  signal rst_IP2Bus_WrAck               : std_logic;
  signal rst_IP2Bus_Error               : std_logic;
  signal rst_Bus2IP_Reset_tmp           : std_logic;
  signal user_Bus2IP_RdCE               : std_logic_vector(USER_NUM_REG-1 downto 0);
  signal user_Bus2IP_WrCE               : std_logic_vector(USER_NUM_REG-1 downto 0);
  signal user_IP2Bus_Data               : std_logic_vector(USER_SLV_DWIDTH-1 downto 0);
  signal user_IP2Bus_RdAck              : std_logic;
  signal user_IP2Bus_WrAck              : std_logic;
  signal user_IP2Bus_Error              : std_logic;

begin

  ------------------------------------------
  -- instantiate axi_lite_ipif
  ------------------------------------------
  AXI_LITE_IPIF_I : entity axi_lite_ipif_v1_01_a.axi_lite_ipif
    generic map
    (
      C_S_AXI_DATA_WIDTH             => IPIF_SLV_DWIDTH,
      C_S_AXI_ADDR_WIDTH             => C_S_AXI_ADDR_WIDTH,
      C_S_AXI_MIN_SIZE               => C_S_AXI_MIN_SIZE,
      C_USE_WSTRB                    => C_USE_WSTRB,
      C_DPHASE_TIMEOUT               => C_DPHASE_TIMEOUT,
      C_ARD_ADDR_RANGE_ARRAY         => IPIF_ARD_ADDR_RANGE_ARRAY,
      C_ARD_NUM_CE_ARRAY             => IPIF_ARD_NUM_CE_ARRAY,
      C_FAMILY                       => C_FAMILY
    )
    port map
    (
      S_AXI_ACLK                     => S_AXI_ACLK,
      S_AXI_ARESETN                  => S_AXI_ARESETN,
      S_AXI_AWADDR                   => S_AXI_AWADDR,
      S_AXI_AWVALID                  => S_AXI_AWVALID,
      S_AXI_WDATA                    => S_AXI_WDATA,
      S_AXI_WSTRB                    => S_AXI_WSTRB,
      S_AXI_WVALID                   => S_AXI_WVALID,
      S_AXI_BREADY                   => S_AXI_BREADY,
      S_AXI_ARADDR                   => S_AXI_ARADDR,
      S_AXI_ARVALID                  => S_AXI_ARVALID,
      S_AXI_RREADY                   => S_AXI_RREADY,
      S_AXI_ARREADY                  => S_AXI_ARREADY,
      S_AXI_RDATA                    => S_AXI_RDATA,
      S_AXI_RRESP                    => S_AXI_RRESP,
      S_AXI_RVALID                   => S_AXI_RVALID,
      S_AXI_WREADY                   => S_AXI_WREADY,
      S_AXI_BRESP                    => S_AXI_BRESP,
      S_AXI_BVALID                   => S_AXI_BVALID,
      S_AXI_AWREADY                  => S_AXI_AWREADY,
      Bus2IP_Clk                     => ipif_Bus2IP_Clk,
      Bus2IP_Resetn                  => ipif_Bus2IP_Resetn,
      Bus2IP_Addr                    => ipif_Bus2IP_Addr,
      Bus2IP_RNW                     => ipif_Bus2IP_RNW,
      Bus2IP_BE                      => ipif_Bus2IP_BE,
      Bus2IP_CS                      => ipif_Bus2IP_CS,
      Bus2IP_RdCE                    => ipif_Bus2IP_RdCE,
      Bus2IP_WrCE                    => ipif_Bus2IP_WrCE,
```

```
    Bus2IP_Data                    => ipif_Bus2IP_Data,
    IP2Bus_WrAck                   => ipif_IP2Bus_WrAck,
    IP2Bus_RdAck                   => ipif_IP2Bus_RdAck,
    IP2Bus_Error                   => ipif_IP2Bus_Error,
    IP2Bus_Data                    => ipif_IP2Bus_Data
  );

------------------------------------------
-- instantiate soft_reset
------------------------------------------
SOFT_RESET_I : entity proc_common_v3_00_a.soft_reset
  generic map
  (
    C_SIPIF_DWIDTH                 => IPIF_SLV_DWIDTH,
    C_RESET_WIDTH                  => RESET_WIDTH
  )
  port map
  (
    Bus2IP_Reset                   => ipif_Bus2IP_Reset,
    Bus2IP_Clk                     => ipif_Bus2IP_Clk,
    Bus2IP_WrCE                    => ipif_Bus2IP_WrCE(RST_CE_INDEX),
    Bus2IP_Data                    => ipif_Bus2IP_Data,
    Bus2IP_BE                      => ipif_Bus2IP_BE,
    Reset2IP_Reset                 => rst_Bus2IP_Reset,
    Reset2Bus_WrAck                => rst_IP2Bus_WrAck,
    Reset2Bus_Error                => rst_IP2Bus_Error,
    Reset2Bus_ToutSup              => open
  );

------------------------------------------
-- instantiate User Logic
------------------------------------------
USER_LOGIC_I : entity tzac_v1_00_a.user_logic
  generic map
  (
    -- MAP USER GENERICS BELOW THIS LINE ---------------
    --USER generics mapped here
    -- MAP USER GENERICS ABOVE THIS LINE ---------------

    C_NUM_REG                      => USER_NUM_REG,
    C_SLV_DWIDTH                   => USER_SLV_DWIDTH
  )
  port map
  (
    -- MAP USER PORTS BELOW THIS LINE ------------------
    S_AXI_AWPROT                   => S_AXI_AWPROT,
    S_AXI_ARPROT                   => S_AXI_ARPROT,
    IRQ                            => IRQ,
    -- MAP USER PORTS ABOVE THIS LINE ------------------

    Bus2IP_Clk                     => ipif_Bus2IP_Clk,
    Bus2IP_Resetn                  => rst_Bus2IP_Reset_tmp,
    Bus2IP_Data                    => ipif_Bus2IP_Data,
    Bus2IP_BE                      => ipif_Bus2IP_BE,
    Bus2IP_RdCE                    => user_Bus2IP_RdCE,
    Bus2IP_WrCE                    => user_Bus2IP_WrCE,
    IP2Bus_Data                    => user_IP2Bus_Data,
    IP2Bus_RdAck                   => user_IP2Bus_RdAck,
    IP2Bus_WrAck                   => user_IP2Bus_WrAck,
    IP2Bus_Error                   => user_IP2Bus_Error
  );

------------------------------------------
-- connect internal signals
------------------------------------------
IP2BUS_DATA_MUX_PROC : process( ipif_Bus2IP_CS, user_IP2Bus_Data ) is
begin

  case ipif_Bus2IP_CS (1 downto 0)  is
    when "01" => ipif_IP2Bus_Data <= user_IP2Bus_Data;
    when "10" => ipif_IP2Bus_Data <= (others => '0');
    when others => ipif_IP2Bus_Data <= (others => '0');
  end case;

end process IP2BUS_DATA_MUX_PROC;

ipif_IP2Bus_WrAck <= user_IP2Bus_WrAck or rst_IP2Bus_WrAck;
ipif_IP2Bus_RdAck <= user_IP2Bus_RdAck;
ipif_IP2Bus_Error <= user_IP2Bus_Error or rst_IP2Bus_Error;

user_Bus2IP_RdCE <= ipif_Bus2IP_RdCE(USER_NUM_REG-1 downto 0);
user_Bus2IP_WrCE <= ipif_Bus2IP_WrCE(USER_NUM_REG-1 downto 0);

ipif_Bus2IP_Reset <= not ipif_Bus2IP_Resetn;
rst_Bus2IP_Reset_tmp <= not rst_Bus2IP_Reset;
end IMP;
```

Note that, for the time being, support for TrustZone exists, but is disabled. We'll modify this later when we're ready for Security Extensions.

- TM Project → Rescan User Repositories
- Tab System Assembly View (Bottom) → Tab Ports (Top) → tzac_0 → IRQ: Connected Port: Pencil Icon → Blue Arrow → Make a note of the number next to tzac_0 in the table on the right → OK

If everything was done correctly, after pressing ctrl + shift + d, your design should resemble figure 2.

PlanAhead

We can now close XPS and return to PlanAhead. We will now create a Top Level HDL file so we can make some lower-level changes later, and then add some design constraints. These are mappings that ensure Xilinx representations of some components match their physical location on the ZedBoard. The constraints we will be adding cover the LEDs, switches, and buttons on the bottom edge of the board (oriented so that the Digilent logo is upright). The following port mappings are printed directly onto the ZedBoard, and can be found online in the schematics PDF in the Redmine wiki (also available from ZedBoard's support site).

- Tab Sources (Top Center) → Design Sources → RC processor_subsystem → Create Top Level HDL
- Tab Sources (Top Center) → RC Constraints → Add Sources

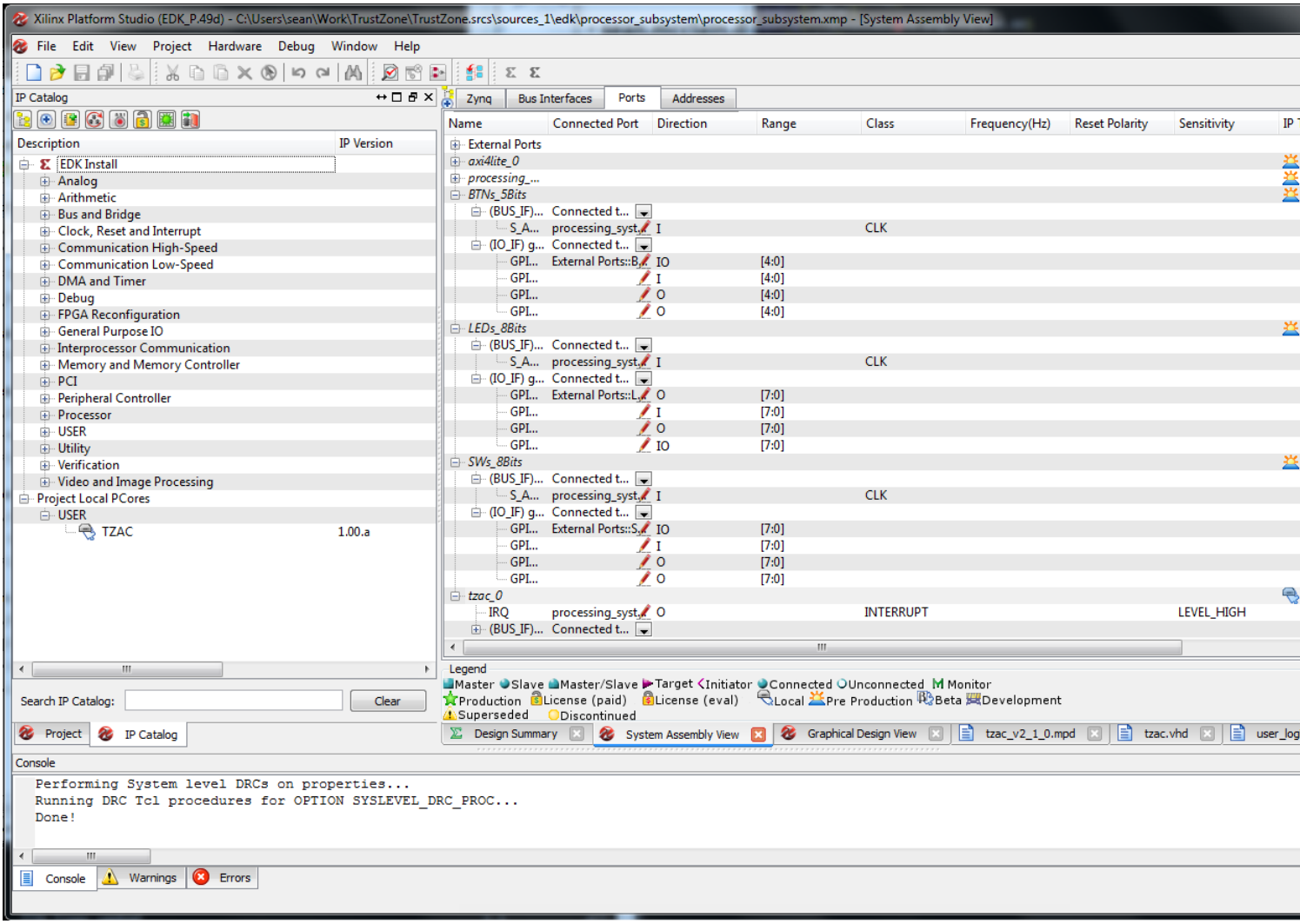

Figure 2: Successfully Configured XPS

- Add or Create Constraints: selected → Next
- Create File
- File name: processing_subsystem_constrs → OK
- Finish
- Tab Sources (Top Center) → Constraints → constrs_1 → DC processing_subsystem_constrs.ucf

You can now edit the contents of processing_subsystem_constrs.ucf in the tab in the upper far right. Populate it with the following:

```
NET LEDs_8Bits_TRI_IO[0] LOC = T22 | IOSTANDARD=LVCMOS33; # "LD0"
NET LEDs_8Bits_TRI_IO[1] LOC = T21 | IOSTANDARD=LVCMOS33; # "LD1"
NET LEDs_8Bits_TRI_IO[2] LOC = U22 | IOSTANDARD=LVCMOS33; # "LD2"
NET LEDs_8Bits_TRI_IO[3] LOC = U21 | IOSTANDARD=LVCMOS33; # "LD3"
NET LEDs_8Bits_TRI_IO[4] LOC = V22 | IOSTANDARD=LVCMOS33; # "LD4"
NET LEDs_8Bits_TRI_IO[5] LOC = W22 | IOSTANDARD=LVCMOS33; # "LD5"
NET LEDs_8Bits_TRI_IO[6] LOC = U19 | IOSTANDARD=LVCMOS33; # "LD6"
NET LEDs_8Bits_TRI_IO[7] LOC = U14 | IOSTANDARD=LVCMOS33; # "LD7"
NET BTNs_5Bits_TRI_IO[0] LOC = P16 | IOSTANDARD=LVCMOS33; # "BTNC"
NET BTNs_5Bits_TRI_IO[1] LOC = R16 | IOSTANDARD=LVCMOS33; # "BTND"
NET BTNs_5Bits_TRI_IO[2] LOC = N15 | IOSTANDARD=LVCMOS33; # "BTNL"
NET BTNs_5Bits_TRI_IO[3] LOC = R18 | IOSTANDARD=LVCMOS33; # "BTNR"
NET BTNs_5Bits_TRI_IO[4] LOC = T18 | IOSTANDARD=LVCMOS33; # "BTNU"
NET SWs_8Bits_TRI_IO[0] LOC = F22 | IOSTANDARD=LVCMOS33; # "SW0"
NET SWs_8Bits_TRI_IO[1] LOC = G22 | IOSTANDARD=LVCMOS33; # "SW1"
NET SWs_8Bits_TRI_IO[2] LOC = H22 | IOSTANDARD=LVCMOS33; # "SW2"
NET SWs_8Bits_TRI_IO[3] LOC = F21 | IOSTANDARD=LVCMOS33; # "SW3"
NET SWs_8Bits_TRI_IO[4] LOC = H19 | IOSTANDARD=LVCMOS33; # "SW4"
NET SWs_8Bits_TRI_IO[5] LOC = H18 | IOSTANDARD=LVCMOS33; # "SW5"
NET SWs_8Bits_TRI_IO[6] LOC = H17 | IOSTANDARD=LVCMOS33; # "SW6"
NET SWs_8Bits_TRI_IO[7] LOC = M15 | IOSTANDARD=LVCMOS33; # "SW7"
```

Note: On every line, if the Base System Builder wizard did not load, you will have to replace "_TRI_IO" with "_GPIO_IO_pin" for all the switch and button lines, and "_GPIO_IO_O_pin" for all the LED lines.

Our hardware design is complete. Compile it and prepare for software design with:

- Tab Flow Navigator (Left) → Program and Debug → Generate Bitstream. This takes a while, and is done in 3 stages: Synthesis, Implementation, and Bitgen. If any stage succeeds, it does not need to be repeated on an error.
- Sources Tab (Top Center) → Design Sources → processor_subsystem_stub - STRUCTURE → RC processor_subsystem_i - processor_subsystem (processor_subsystem.xmp) → Export Hardware for SDK... → Launch SDK: checked → OK

Note: If you see any of the following error messages, they are safe to ignore:

```
[Constraints 18-5]
  Cannot loc instance 'processing_system7_0_PS_PORB_IBUF' at site B5, Site location is not valid
  ["C:/TrustZone/TrustZone.srcs/sources_1/edk/processor_subsystem/implementation/processor_subsystem_processing_system7_0_wrapper.ncf":157]
[Constraints 18-5]
  Cannot loc instance 'processing_system7_0_PS_SRSTB_IBUF' at site C9, Site location is not valid
  ["C:/TrustZone/TrustZone.srcs/sources_1/edk/processor_subsystem/implementation/processor_subsystem_processing_system7_0_wrapper.ncf":158]
[Constraints 18-5]
  Cannot loc instance 'processing_system7_0_PS_CLK_IBUF' at site F7, Site location is not valid
  ["C:/TrustZone/TrustZone.srcs/sources_1/edk/processor_subsystem/implementation/processor_subsystem_processing_system7_0_wrapper.ncf":159]
```

If everything was done correctly, and you set up your Messages tab to only show critical warnings, your PlanAhead window should resemble figure 3.

2 Software Design

The remainder of the design will be done in the Xilinx Software Development Kit (XSDK).

XSDK

We will first need to configure the ZedBoard so that we can program it. To make sure that it is properly configured, we will use a simple program which will access some of the ZedBoard's peripheral hardware:

- TM File → New → Application Project
- Project name: normal_world → finish
- Tab Project Explorer (left) → normal_world → src → DC helloworld.c
- Tab helloworld.c (Top) → Replace the contents of helloworld.c with the following:

```
#include <stdio.h>
#include "platform.h"
#include "xil_types.h"
#include "xgpio.h"
#include "xparameters.h"
#include "tzac.h"
#include "xscugic.h"

#define LED_CHANNEL 1
#define CYCLE_RATE 76 /* calculated by hand, cycles/us */
#define LED_DELAY(x) (x * CYCLE_RATE) /* cycles in x us */
typedef enum _LED_DIR
{
    LED_DIR_UP,
```

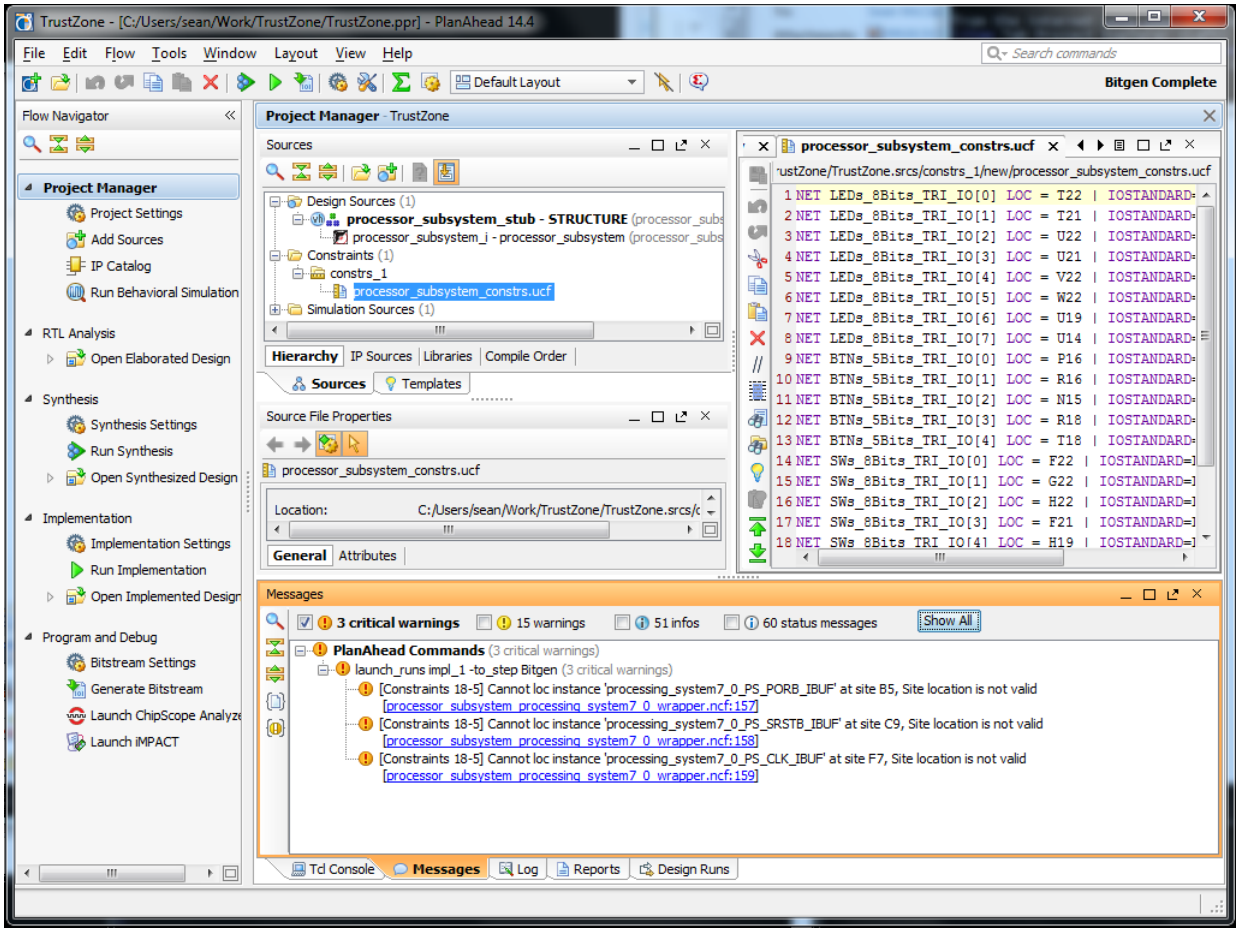

Figure 3: Successfully Generated Bitstream

```
    LED_DIR_DOWN
} LED_DIR;

XGpio Gpio; /* The Instance of the GPIO Driver */
XScuGic InterruptController; /* Instance of the Interrupt Controller */
static XScuGic_Config *GicConfig;/* The configuration parameters of the
                                    controller */
volatile int irq_triggered = 0;

void irq_handler(void *callbackarg)
{
    static char *buff = "contents: 0x00000000\r\n";
    static size_t len = 24;

    snprintf (
     buff, len, "contents: 0x%08X\r\n", (unsigned) TZAC_mReadSlaveReg0()
     );
    print(buff);

    TZAC_mWriteSlaveReg1("0x00000002");
}

int init()
{
    void *callbackarg = &InterruptController;

    init_platform();
    TZAC_mReset();

    /* AXI Led GPIO Initialization */
    if(XST_SUCCESS != XGpio_Initialize(&Gpio, XPAR_LEDS_8BITS_DEVICE_ID))
    {
         return XST_FAILURE;
    }

    /* Initialize the interrupt controller driver so that it is ready to use. */
    Xil_ExceptionInit();
    if ( NULL == (GicConfig = XScuGic_LookupConfig(XPAR_PS7_SCUGIC_0_DEVICE_ID)) )
    {
        return XST_FAILURE;
    }
    if (XST_SUCCESS != (XScuGic_CfgInitialize (
            &InterruptController, GicConfig, GicConfig->CpuBaseAddress
            )))
    {
        return XST_FAILURE;
    }

    /* Connect the interrupt controller interrupt handler to the hardware
       interrupt handling logic in the ARM processor. */
    Xil_ExceptionRegisterHandler (
            XIL_EXCEPTION_ID_IRQ_INT,
            (Xil_ExceptionHandler) XScuGic_InterruptHandler,
            &InterruptController
            );

    /* Enable interrupts in the ARM */
    Xil_ExceptionEnable();

    /* Connect a device driver handler that will be called when an
       interrupt for the device occurs, the device driver handler performs
```

```
       the specific interrupt processing for the device */
    if (XST_SUCCESS != XScuGic_Connect (
          &InterruptController, TZAC_INTERRUPT_ID, irq_handler, callbackarg
          ))
    {
        return XST_FAILURE;
    }

    /* Enable the interrupt for the device and then cause (simulate) an
       interrupt so the handlers will be called */
    XScuGic_Enable (&InterruptController, TZAC_INTERRUPT_ID);

    return 0;
}

/**
 * Cycles an LED once and waits the specified delay
 */
int chase_leds()
{
    static LED_DIR dir = LED_DIR_UP;
    static u32 pattern = 0x00000001;
    u32 i = 0;

    XGpio_DiscreteWrite(&Gpio, (unsigned) LED_CHANNEL, pattern);

    pattern = dir == LED_DIR_UP ? pattern << 1 : pattern >> 1;
    dir = pattern >= 0x00000080
        ? LED_DIR_DOWN
        : pattern <= 0x00000001
        ? LED_DIR_UP
        : dir;
    if (pattern >= 0x00000080)
    {
        TZAC_mWriteSlaveReg1(0x00000000);
        snprintf (
                buff, len, "contents: 0x%08X\r\n",
                (unsigned) TZAC_mReadSlaveReg1()
                );
        print(buff);
    }

    while (++i < LED_DELAY(100000));
    return 0;
}

int main()
{
    if (init())
    {
        return 1;
    }

    while (! (irq_triggered || chase_leds()) );

    cleanup_platform();
    return 0;
}
```

- Tab Project Explorer (left) → normal_world → RC src → Import...
- Tab (Center) → General → File System → Next
- Browse → (your working directory) → TrustZone → TrustZone.srcs → sources_1 → edk → drivers → tzac_v1_00_a → src → OK
- Tab (Right) → tzac.h: checked → Finish
- Tab Project Explorer (left) → normal_world → src → DC tzac.h
- Tab tzac.h (Center) → replace the contents of tzac.h with the following:

```
/*****************************************************************************
* Filename:          C:\Users\sean\Work\TrustZone\TrustZone.srcs\sources_1\edk\processor_subsystem/drivers/tzac_v1_00_a/src/tzac.h
* Version:           1.00.a
* Description:       tzac Driver Header File
* Date:              Fri Sep 20 09:10:42 2013 (by Create and Import Peripheral Wizard)
*****************************************************************************/

#ifndef TZAC_H
#define TZAC_H

/***************************** Include Files *******************************/

#include "xbasic_types.h"
#include "xstatus.h"
#include "xil_io.h"

/************************** Constant Definitions ***************************/

#define TZAC_BASE_OFFSET 0x6E800000
#define TZAC_INTERRUPT_ID 91

/**
 * User Logic Slave Space Offsets
 * -- SLV_REG0 : user logic slave module register 0
 * -- SLV_REG1 : user logic slave module register 1
 * -- SLV_REG2 : user logic slave module register 2
 */
#define TZAC_USER_SLV_SPACE_OFFSET (0x00000000)
#define TZAC_SLV_REG0_OFFSET (TZAC_USER_SLV_SPACE_OFFSET + 0x00000000)
#define TZAC_SLV_REG1_OFFSET (TZAC_USER_SLV_SPACE_OFFSET + 0x00000004)
#define TZAC_SLV_REG2_OFFSET (TZAC_USER_SLV_SPACE_OFFSET + 0x00000008)

/**
 * Software Reset Space Register Offsets
 * -- RST : software reset register
 */
#define TZAC_SOFT_RST_SPACE_OFFSET (0x00000100)
#define TZAC_RST_REG_OFFSET (TZAC_SOFT_RST_SPACE_OFFSET + 0x00000000)

/**
 * Software Reset Masks
 * -- SOFT_RESET : software reset
 */
#define SOFT_RESET (0x0000000A)

/************************** Type Definitions ***************************/
```

```
/****************** Macros (Inline Functions) Definitions *******************/

/**
 *
 * Write a value to a TZAC register. A 32 bit write is performed.
 * If the component is implemented in a smaller width, only the least
 * significant data is written.
 *
 * @param   Reg is the register to write to.
 * @param   Data is the data written to the register.
 *
 * @return  None.
 *
 * @note
 * C-style signature:
 *  void TZAC_mWriteReg(Xuint32 BaseAddress, unsigned RegOffset, Xuint32 Data)
 *
 */
#define TZAC_mWriteReg(Reg, Data) \
  Xil_Out32((TZAC_BASE_OFFSET) + (Reg * 0x04), (Xuint32)(Data))

/**
 *
 * Read a value from a TZAC register. A 32 bit read is performed.
 * If the component is implemented in a smaller width, only the least
 * significant data is read from the register. The most significant data
 * will be read as 0.
 *
 * @param   Reg is the register to write to.
 *
 * @return  Data is the data from the register.
 *
 * @note
 * C-style signature:
 *  Xuint32 TZAC_mReadReg(Xuint32 BaseAddress, unsigned RegOffset)
 *
 */
#define TZAC_mReadReg(Reg) \
  Xil_In32((TZAC_BASE_OFFSET) + (Reg * 0x04))


/**
 *
 * Write/Read 32 bit value to/from TZAC user logic slave registers.
 *
 * @param   Value is the data written to the register.
 *
 * @return  Data is the data from the user logic slave register.
 *
 * @note
 * C-style signature:
 *  void TZAC_mWriteSlaveRegn(Xuint32 BaseAddress, unsigned RegOffset, Xuint32 Value)
 *  Xuint32 TZAC_mReadSlaveRegn(Xuint32 BaseAddress, unsigned RegOffset)
 *
 */
#define TZAC_mWriteSlaveReg0(Value) \
  Xil_Out32((TZAC_BASE_OFFSET) + (TZAC_SLV_REG0_OFFSET), (Xuint32)(Value))
#define TZAC_mWriteSlaveReg1(Value) \
  Xil_Out32((TZAC_BASE_OFFSET) + (TZAC_SLV_REG1_OFFSET), (Xuint32)(Value))
#define TZAC_mWriteSlaveReg2(Value) \
  Xil_Out32((TZAC_BASE_OFFSET) + (TZAC_SLV_REG2_OFFSET), (Xuint32)(Value))

#define TZAC_mReadSlaveReg0() \
  Xil_In32((TZAC_BASE_OFFSET) + (TZAC_SLV_REG0_OFFSET))
#define TZAC_mReadSlaveReg1() \
  Xil_In32((TZAC_BASE_OFFSET) + (TZAC_SLV_REG1_OFFSET))
#define TZAC_mReadSlaveReg2() \
  Xil_In32((TZAC_BASE_OFFSET) + (TZAC_SLV_REG2_OFFSET))

/**
 *
 * Reset TZAC via software.
 *
 * @return  None.
 *
 * @note
 * C-style signature:
 *  void TZAC_mReset(Xuint32 BaseAddress)
 *
 */
#define TZAC_mReset() \
  Xil_Out32((TZAC_BASE_OFFSET)+(TZAC_RST_REG_OFFSET), SOFT_RESET)

/************************** Function Prototypes *****************************/


/**
 *
 * Run a self-test on the driver/device. Note this may be a destructive test if
 * resets of the device are performed.
 *
 * If the hardware system is not built correctly, this function may never
 * return to the caller.
 *
 * @param   baseaddr_p is the base address of the TZAC instance to be worked on.
 *
 * @return
 *
 *    - XST_SUCCESS   if all self-test code passed
 *    - XST_FAILURE   if any self-test code failed
 *
 * @note    Caching must be turned off for this function to work.
 * @note    Self test may fail if data memory and device are not on the same bus.
 *
 */
XStatus TZAC_SelfTest(void * baseaddr_p);
/**
*  Defines the number of registers available for read and write*/
#define TEST_AXI_LITE_USER_NUM_REG 3


#endif /** TZAC_H */
```

- Tab tzac.h (Center) → Toward the beginning after it says "Constant Definitions" (line 19), replace the value of TZAC_BASE_OFFSET with the address that was in XPS: Tab System Assembly View (Bottom) : Tab Addresses (Top) : tzac_0 : Base Address, when you were working in XPS earlier. If you did not make a note of this address, you can get into XPS from PlanAhead with Tab Sources (Top Center) → Design Sources → processor_subsystem_stub - STRUCTURE → DC processor_subsystem_i - processor_subsystem .

- Tab tzac.h (Center) → Toward the beginning after it says "Constant Definitions" (line 20), replace the value of TZAC_INTERRUPT_ID with the value you were asked to remember just after clicking the Blue arrow while hooking up the tzac's interrupt port in the XPS section earlier. 91 is a reasonable value. If you do not remember this value, you can find it in XPS → Tab System Assembly View (Bottom) → Tab Ports (Top) → tzac_0 → IRQ : Connected Port → click the pencil.
- Tab Project Explorer (left) → RC normal_world → Generate Linker Script
- Generate
- If a popup saying "Linker Script Already Exists!" appears, Yes
- Refer to the instructions at http://redmine.ais/projects/trustpwn/wiki/Wiki#Accessing-the-ZedBoard and put the ZedBoard in the JTAG configuration
- TM Xilinx Tools → Program FPGA
- (assuming your working directory is C:), Bitstream:

  `C:\TrustZone\TrustZone.runs\impl_1\processor_subsystem_stub.bit`

  → Program
- Tab Project Explorer (Left) → RC normal_world → Run As → Run Configurations...
- Tab (Left) → DC Xilinx C/C++ ELF → Run

After a little while, you should see "system start up" in your terminal emulator window, and LEDs 0-7 should start scanning back and forth, as shown in figure 4. If you change helloworld line 105 from "TZAC_mWriteSlaveReg1(0x00000000);" to "TZAC_mWriteSlaveReg1(0xFFFFFFFF);", you should see the LED make a half cycle and stop, and you should see "interrupt acknowledged" and "contents: 0xFFFFFFFF" in the terminal emulator. This lets you know that your hardware interrupt from your peripheral worked properly.

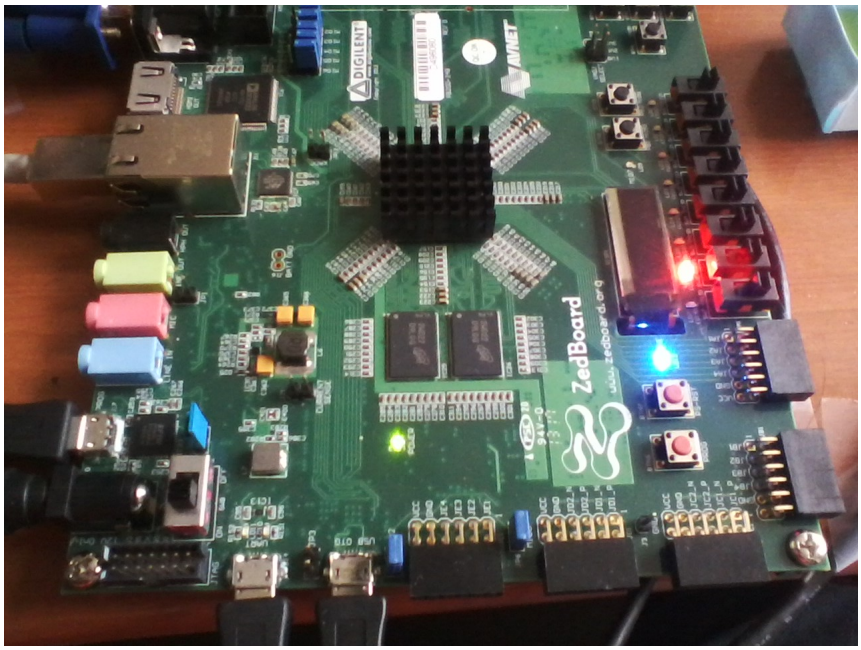

Figure 4: Successfully Configured Board

Next we will create the First-Stage Boot Loader (FSBL). This is the very first program that the ARM chip will run. We will for the time being configure this to run our LED test program.

- TM File → New → Application Project
- Project name: fsbl → Tab Target Software (Bottom)
  → Board Support Package: Use existing: normal_world_bsp → Next
- Tab Available Templates (Left) → Zynq FSBL → Finish
- Optional, since we will get rid of normal_world later

  - Tab Project Explorer (Left) → RC normal_world_bsp → Rename
  - New name: trustzone_bsp → OK
  - Tab Project Explorer (Left) → RC fsbl → Change Referenced BSP
  - trustzone_bsp → OK
  - Tab Project Explorer (Left) → RC normal_world → Change Referenced BSP
  - trustzone_bsp → OK
- Tab Project Explorer (Left) → fsbl → src → DC lscript.ld
- Tab Project Explorer (Left) → processor_subsystem_hw_platform → DC system.xml
- Tab Summary (Center Bottom) → Tab lscript.ld (Center Top) → Add Memory... (may have to scroll right to see this button)
- Tab Summary (Center Bottom) → Tab system.xml (Center Top) → Address Map for processor ps7_cortexa9_0 : ps7_ddr_0 : Copy these two hexadecimal addresses down. We will refer to them as DDR_START_ADDRESS and DDR_END_ADDRESS.
- Tab Summary (Center Bottom) → Tab lscript.ld (Center Top) → Available Memory Regions: newMemory2: change this to ps7_ddr_0_S_AXI_BASEADDR
- Note: you can copy these next two values from the lscript.ld file for normal_world. They are being pulled from system.xml to avoid name confusion, for instructive purposes, and so you will not need to compile a "hello world" package first in the future.
- Tab Summary (Center Bottom) → Tab lscript.ld (Center Top) → Available Memory Regions: ps7_ddr_0_S_AXI_BASEADDR : Base Address : DDR_START_ADDRESS
- Tab Summary (Center Bottom) → Tab lscript.ld (Center Top) → Available Memory Regions: ps7_ddr_0_S_AXI_BASEADDR : Size: DDR_END_ADDRESS - DDR_START_ADDRESS + 1. Example: if DDR_END_ADDRESS = 0x1FFFFFFF and DDR_START_ADDRESS = 0x00100000, then you would fill in 0x1FF0000.
- TM Xilinx Tools → Create Zynq Boot Image
- Bif file : Create a new bif file → FSBL elf : C:/TrustZone/TrustZone.sdk/SDK/SDK_Export/fsbl/Debug/fsbl.elf → Add → C:/TrustZone/TrustZone.runs/impl_1/processor_subsystem_stub.bit → Add → C:/TrustZone/TrustZone.sdk/SDK/S → Ensure the order fsbl.elf, then processor_subsystem_stub.bit, then normal_world.elf → Tab (Bottom) → Browse → Expand your working directory → TrustZone → Make New Folder → TrustZone.bootgen → OK → Create Image
- Refer to the instructions at http://redmine.ais/projects/trustpwn/wiki/Wiki#Accessing-the-ZedBoard and put the ZedBoard in the Quad-SPI configuration
- TM Xilinx Tools → Program Flash
- Image File: C:/TrustZone/TrustZone.bootgen/normal_world.mcs → Offset: 0x00000000 → Verify after flash: checked → Program (this takes a while)
- (Optional) Physically unplug your JTAG cable from the ZedBoard
- Physically power cycle the ZedBoard

If everything worked correctly, you should again get the output you received in figure 4. Now we will attempt to shift responsibility for the LED scanning back-and-forth to the second processor.