AUTOMATIC PET FEEDER

Bilkent University Electrical and Electronics Engineering Department EEE102 Term Project

Can Ilgar Kurnaz

22202248
Section 3

Table of Contents

Objective

Aim of this project was to design a contraption that can automatically add food and water to determined food and water containers (Figure 1.1) whenever container level was below some predetermined threshold. This contraption should also give warning signals whenever the storage level is low or dirty.

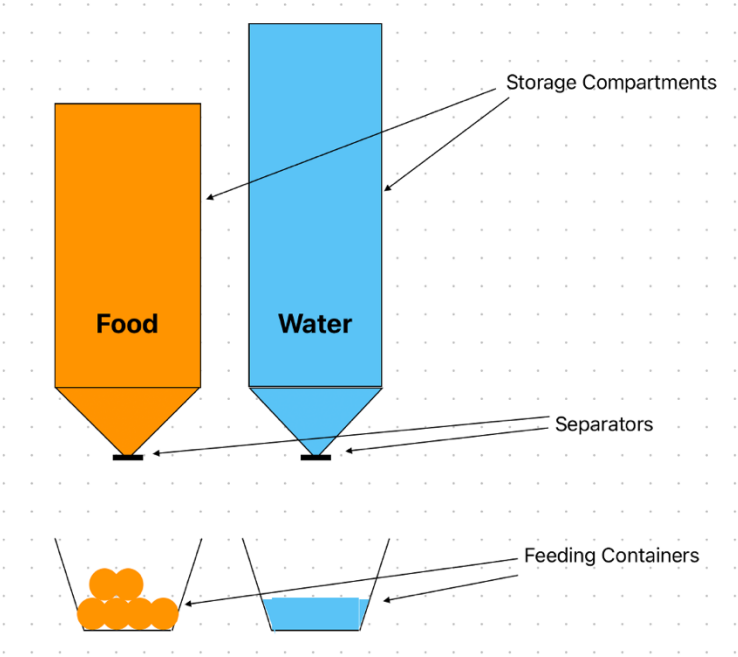


Figure 1.1: Very Basic Schematic of The Dispenser

Components

Various components were used throughout the project to achieve the desired outcome. Namely one BASYS 3 FPGA, one servo motor, one infrared range sensor, two water level sensors, one motor driver, one water pump, and one force sensing resistor were used.

Servo Motor

One MG995 servo motor (Figure 2.1) was used to control the separation between food storage and food container. This servo is controlled with a 50 Hz PWM signal, and a 1ms-2ms duty cycle.

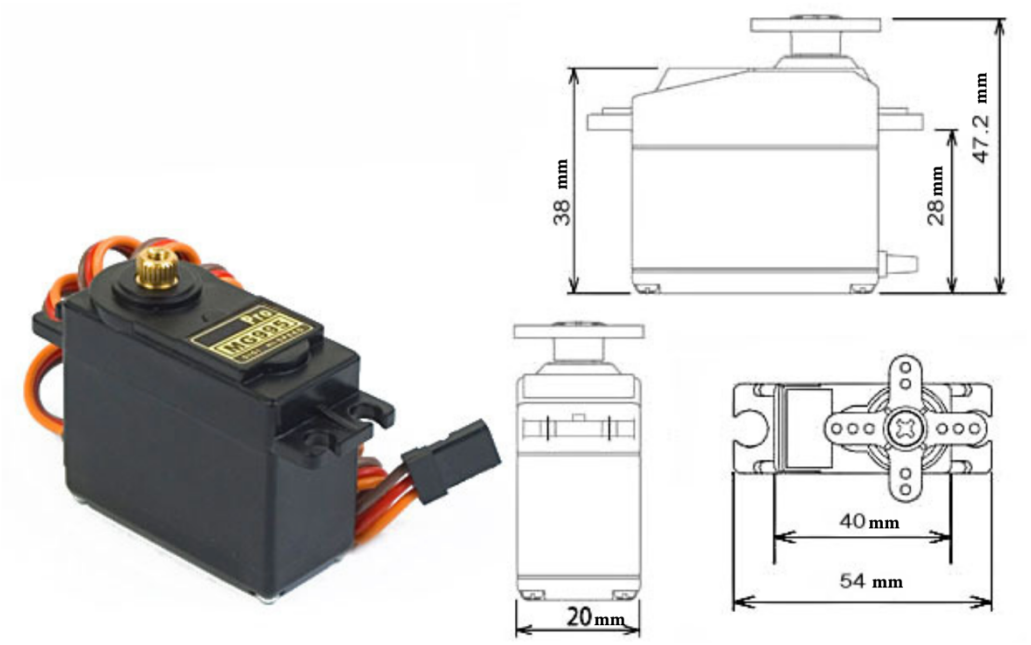


Figure 2.1: MG995 Servo Motor

Infrared Range Sensor

One E18-D80NK-N IR sensor switch (Figure 2.2) was used to keep track of food in the food container. This sensor gives logic 0 output when there is no object in range and gives logic 1 output whenever there is an object in range. Range of the sensor is tuned with a turning screw located at the back of the sensor.

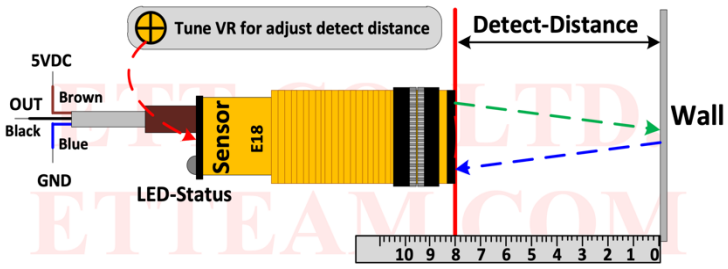


Figure 2.2: Infrared Range Sensor

Water Level Sensor

Two water level sensors (Figure 2.3) were used to determine water level on both the container and the storage compartments. These sensors give analog output depending on the water level, they were used with a resistive voltage divider to convert output voltage levels to wanted levels so that BASYS 3 can register them as logic 0 or logic 1.

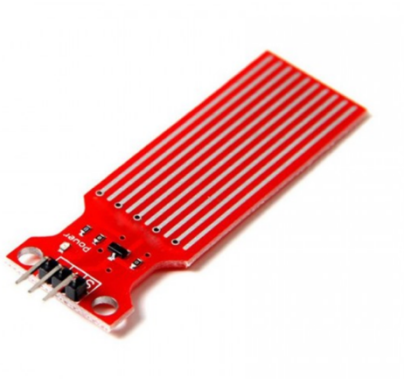


Figure 2.3: Water Level Sensor

Motor Driver

One motor driver (Figure 2.4) with L298N voltage regulator was used to drive the water pump with 3.3 V outputs from BASYS 3 FPGA. I connected the BASYS 3 pins to in1 and in2 and took outputs adjusted to 12V out1 and out2 and connected the outputs to the water pump.

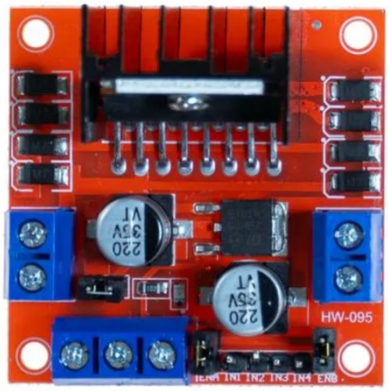


Figure 2.4: Motor Driver

Water Pump

One water pump (Figure 2.5) was used when making the transfer mechanism between water container and water storage.

Figure 2.5: Water Pump

Force Sensing Resistor

One Force sensing sensor (Figure 2.6) was used to keep track of the food level in the feeding container. This sensor changers its resistivity (Figure 2.7) depending on the force applied to the sensor. It was used with a 3.3V voltage supply and with a resistive voltage divider to achieve voltage levels in between 0 and 1 volts which can be read with A/D converter chip on BASYS 3 board.

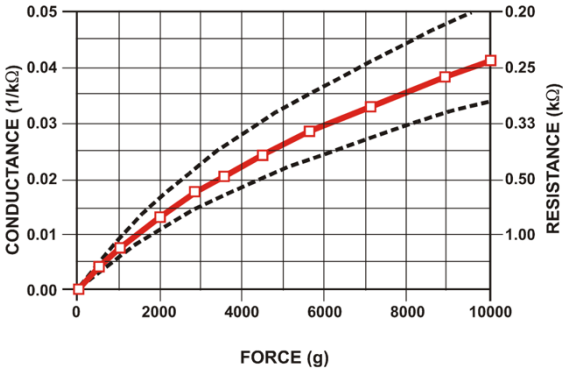


Figure 2.7: Force/Resistance Curve

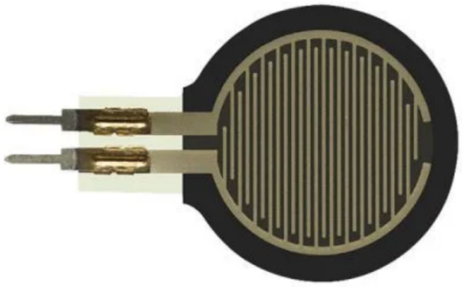


Figure 2.6: FSR

Methodology

First step in design process was to design the algorithm which specifies when separator is removed and water pump is turned on. Second step was to write sensor interfaces. Interface for the IR sensor and for water level sensors was relatively simple and they were treated as digital sensors with 1 or 0 outputs with right physical adjustments. But the interface for the force sensing resistor included an A/D converter written in Vivado specifically tuned for BASYS 3 FPGA. Then the A/D conversion result was converted to logic 0 or logic 1 depending on a experimentally set threshold. Third process was writing the PWM (Pulse Width Modulation) code for controlling the servo. Fourth process was writing the water motor control code.

Design Specifications

The design (Figure 3.1) had one main module, 3 submodules A/D converter, PWM, Water Motor Driver under the main module and had 2 submodules under A/D converter to tune the A/D converter for BASYS 3.

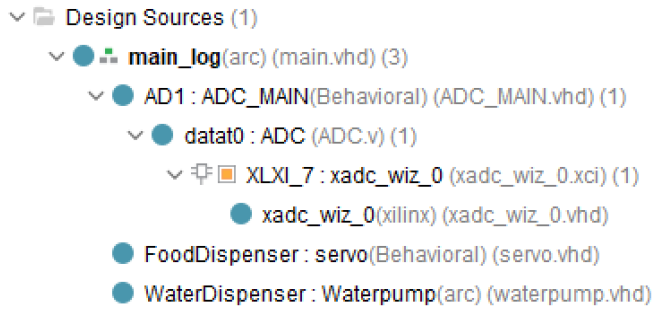


Figure 3.1: Design Hierarchy

Main module (Figure 3.4) had two water container level inputs one positive, one negative for A/D converter. One water storage level input, one food storage input, and one food container input. Inputs signals are converted to logic 0 and 1 and given as outputs in this module. Also it had counters to give warning whenever container was dirty, these counters will give warning every two days to make user clean the storages, and every day to make user clean feeding containers.

A/D converter (Figure 3.2) was tuned with ADC wizard of Vivado and used to convert the input from force sensing resistor to a 16-bit digital input.

Servo driver (Figure 3.3) was written with a counter which took values in between 1 and 2,000,000 and increased by 1 with every clock cycle of internal clock of BASYS 3 the output takes 1 if this value is in between 1 and 199,999 or when it is in between 1 and 249,999 which corresponds to 1 ms and 1.5 ms duty cycle respectively and was used to control the servo between 0 and 90 degrees.

Water pump driver gives output ‘10’ when pump needs to be driven and takes ‘00’ when pump needs to be idle.

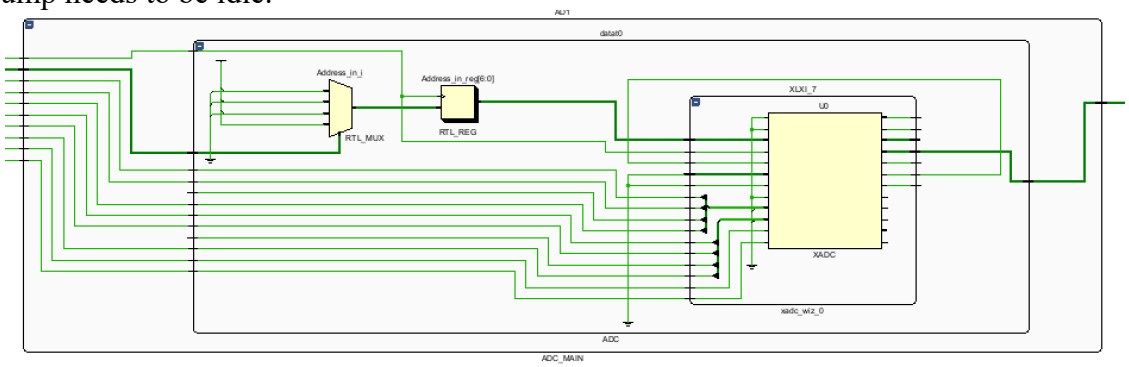


Figure 3.2: RTL Schematic of A/D converter

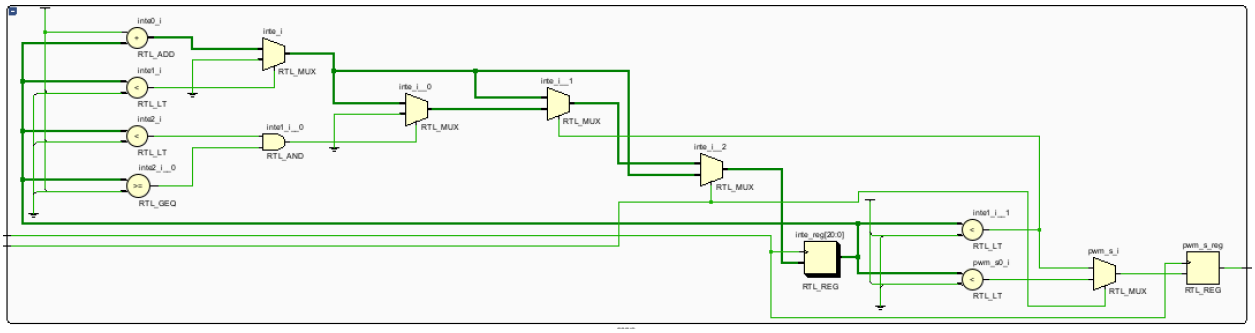


Figure 3.3: RTL Schematic of Servo Driver

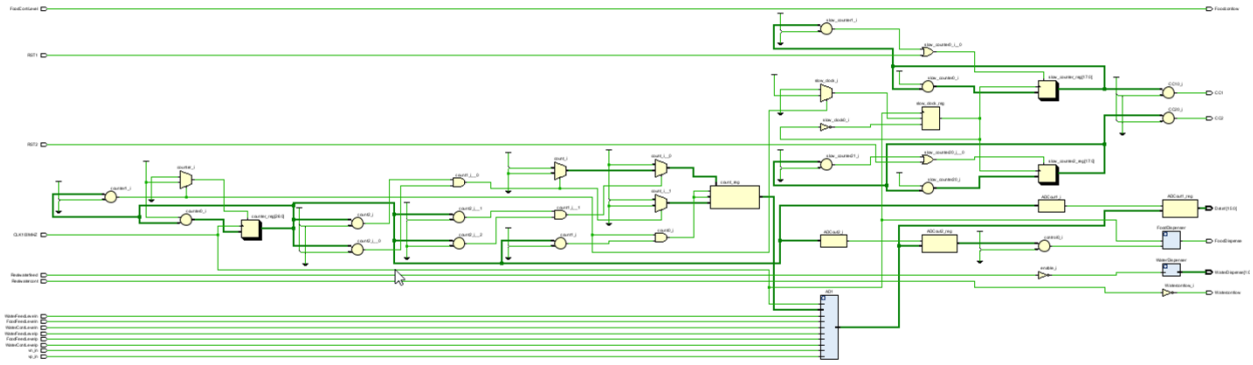
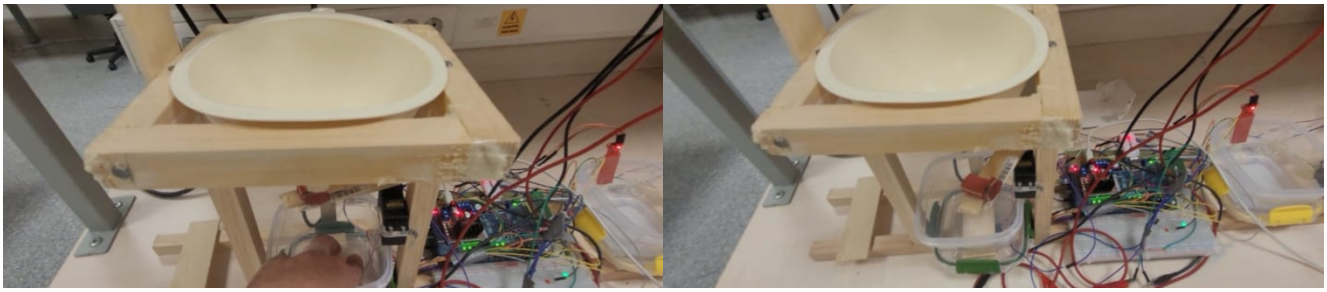


Figure 3.4: RTL Schematic of Whole Design

After this process circuit was connected to the contraption with respect to constraint files.

Results

Food separator will be closed whenever there is enough force on the food container (Figure 4.1 and Figure 4.2).



Figures 4.1 and 4.2: Food Separator

Water level is always kept at stable level (Figure 4.3).

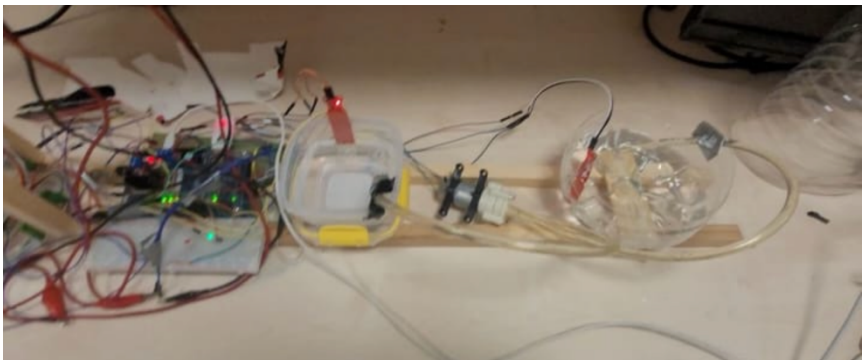


Figure 4.3: Water Level Stable

When food storage is Low, the green LED will turn on (Figure 4.4 and Figure 4.5).

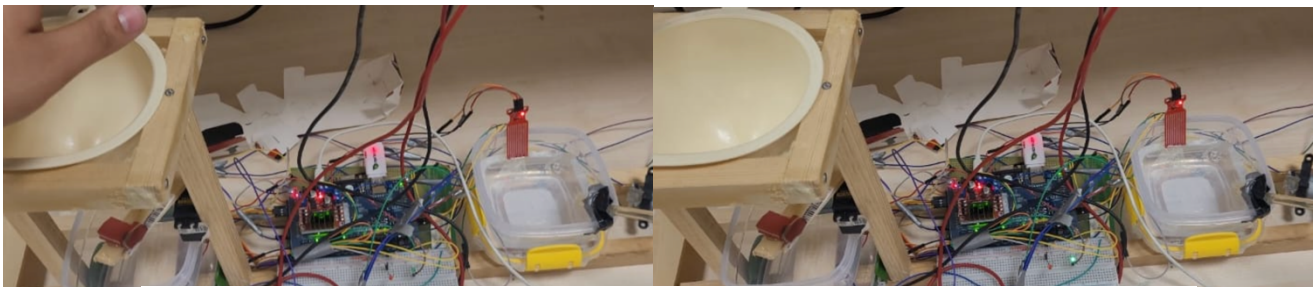


Figure 4.4 and Figure 4.5: Food Level Low Warning

When water storage is low the red LED will turn on (Figure 4.6 and Figure 4.7). On Figure 4.6 the sensor is slightly pulled upwards.

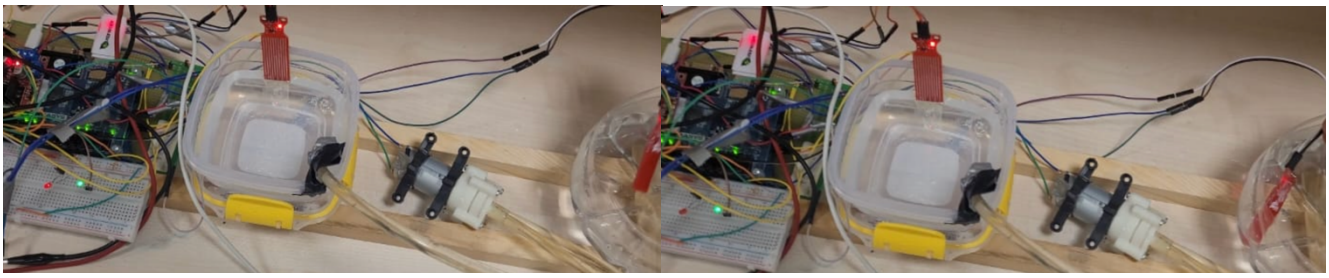


Figure 4.6 and Figure 4.7: Water Level Low Warning

Important Note: Cleaning warning could not be tested because the the tester couldn't keep BASYS3 on power for two days as BASYS 3 wipes all memory when power is turned off.

Conclusion

Aim of this project was to somehow automate the feeding process for pets. I was successful at implementing this idea on BASYS 3 FPGA.

In this project I have learned to write VHDL interface for different types of external components and sensors. Most challenging part of the project was learning how to implement a A/D converter on BASYS 3. When implementing this I encountered many errors caused by analog systems. I had to debounce the analog signals to avoid false readings caused by environmental causes. I had to slow down the reading resolution of the program to avoid faulty readings. Second hardest part in this process was learning how VHDL code works. I used different online sources to learn VHDL from basically 0 to somewhat functional knowledge.

This project was very helpful at teaching me how digital and analog systems work, how servos functions, how VHDL works, how BASYS 3 works. So overall this project was very successful at helping me learn EEE102 concepts and implement them on real world.

References

Digilent. (n.d.). Basys 3 XADC Demo - Digilent Reference. Retrieved December 30, 2023, from https://digilent.com/reference/programmable-logic/basys-3/demos/xadc.

Direnc.net. (n.d.). Arduino Sıvı Seviye Sensörü. Retrieved December 30, 2023, from https://www.direnc.net/arduino-sivi-seviye-sensoru-funduino.

ETT. (n.d.). E18-D80NK-N datasheet. Retrieved December 30, 2023, from https://datasheetspdf.com/pdf-file/1311838/ETT/E18-D80NK-N/1

Marlin P. Jones & Assoc., Inc. (n.d.). MG995 Datasheet [PDF]. Retrieved December 30, 2023, from https://pdf1.alldatasheet.com/datasheet-pdf/view/1132435/ETC2/MG995.html.

Pololu. (n.d.). FSR® Integration Guide & Evaluation Parts Catalog With Suggested Electrical Interfaces [PDF file]. Retrieved from https://www.pololu.com/file/download/fsr_datasheet.pdf?file_id=0J383

Robotistan. (n.d.). Su Seviyesi / Yağmur Sensörü - Water Level / Rain Sensor. Retrieved December 30, 2023, from https://www.robotistan.com/su-seviyesi-yagmur-sensoru-water-level-rain-sensor&#8203

Robiduck. (n.d.). Diyaframlı su pompası (12V Su Motoru). Retrieved December 30, 2023, from https://www.robiduck.com/urun/diyaframli-su-pompasi-12v-su-motoru

Appendix

main.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.std_logic_arith.all;
entity main_log is
port (
WaterFeedLevelp, WaterFeedLeveln, FoodFeedLevelp, FoodFeedLeveln :in Std_logic;
WaterContLevelp, WaterContLeveln, FoodContLevel, CLK100MHZ, RST1,RST2: in std_logic;
FoodDispense: out std_logic;
WaterDispense: out std_logic_vector(1 downto 0);
Watercontlow, Foodcontlow: out std_logic;
Realwatercont, Realwaterfeed: in std_logic;
CC1, CC2: out std_logic;
vp_in, vn_in: in std_logic;
Data1: out std_logic_vector(15 downto 0)
);
end main_log;

----------------------------------------------------------------------------
architecture arc of main_log is


signal Cycles1, Cycles2: integer range 0 to 4; --Signal initilasation
signal enable, control:  std_logic;
signal ADCout1, ADCout2, ADCout3,ADO: std_logic_vector(15 downto 0);
signal CleanCyc1, CleanCyc2,WCL: std_logic;
signal count: std_logic_vector(1 downto 0):= "00";
signal counter: integer range 0 to 1_000_000_00 := 0;
signal slow_clock: std_logic;
signal slow_counter: integer range 0 to 200_000 := 0;
signal slow_counter2: integer range 0 to 200_000 := 0;
signal enable_ns, control_ns, WCL_ns: std_logic;
----------------------------------------------------------------------------


component Waterpump --Comp init start
   port(enable: in std_logic;
       drive: out std_logic_vector(1 downto 0)
   );
end component;
```

```
component Servo
 Port (control: in std_logic;
      CLK100MHZ: in std_logic;
      pwm: out std_logic  );
end component;

component ADC_MAIN
Port (CLK100MHZ,vauxp6,vauxn6,vauxp7,vauxn7,vauxp15,vauxn15,vp_in,vn_in: in std_logic;
     Data1: out std_logic_vector(15 downto 0);
     sw: in std_logic_vector(1 downto 0)
       );
end component;


begin --Port Maps for AD conversion outputs




-----------------------------------------------------------------------------------------------
AD1: ADC_MAIN port map (CLK100MHZ => CLK100MHZ, vauxp6 => WaterFeedLevelp, vauxn6
=>WaterFeedLeveln,
                  vauxp7 => FoodFeedLevelp, vauxn7 => FoodFeedLeveln, vauxp15 =>
WaterContLevelp,
                  vauxn15 => WaterContLeveln, vp_in => vp_in, vn_in => vn_in, sw => count,
                  Data1 => ADO);
--AD4: ADC_MAIN port map (CLK100MHZ => CLK100MHZ, vauxp6 => WaterFeedLevelp, vauxn6
=>WaterFeedLeveln,
--                  vauxp7 => FoodFeedLevelp, vauxn7 => FoodFeedLeveln, vauxp15 =>
WaterContLevelp,
--                  vauxn15 => WaterContLeveln, vp_in => vp_in, vn_in => vn_in, sw => "10",
--                  Data1 => ADCout1 );
--AD2: ADC_MAIN port map (CLK100MHZ => CLK100MHZ, vauxp6 => WaterFeedLevelp, vauxn6
=>WaterFeedLeveln,
--                  vauxp7 => FoodFeedLevelp, vauxn7 => FoodFeedLeveln, vauxp15 =>
WaterContLevelp,
--                  vauxn15 => WaterContLeveln, vp_in => vp_in, vn_in => vn_in, sw => "10",
--                  Data1 => ADCout2  );

--AD3: ADC_MAIN port map (CLK100MHZ => CLK100MHZ, vauxp6 => WaterFeedLevelp, vauxn6
=>WaterFeedLeveln,
```

```
--                  vauxp7 => FoodFeedLevelp, vauxn7 => FoodFeedLeveln, vauxp15 =>
WaterContLevelp,
--                  vauxn15 => WaterContLeveln, vp_in => vp_in, vn_in => vn_in, sw => "11",
--                  Data1 => ADCout3  );

FoodDispenser: Servo port map(control => control, CLK100MHZ => CLK100MHZ, pwm =>
FoodDispense);

WaterDispenser: Waterpump port map(enable => enable, drive=> WaterDispense);


process(CLK100MHZ) begin
if rising_edge(CLK100MHZ) then
   if counter < 1_000_000_00 then
      counter <= counter + 1;
   else counter <= 0;slow_clock <= not(slow_clock); end if;


end if;

end process;

process(counter) begin
if counter < 300_000_00 and counter > 0 then
   count <= "00";
--    if ADO < "0010-0000-0000-0000"  and counter = 200_000 then
--       enable <= '1'; else enable <= '0'; end if;
elsif counter < 600_000_00 and counter > 300_000_00 then
   count <= "01";
----       if ADO > "0010-0111-1111-1111" and counter = 500_000 then
----       control <= '1'; else control <= '0'; end if;
elsif counter < 1_000_000_00 and counter > 600_000_00 then
   count <= "11";
--        if ADO < "0010-0000-0000-0000" and counter = 800_000 then
--        WCL <= '1'; else WCL <= '0'; end if;

end if;


if counter = 200_000_00 then ADCout1 <= ADO; end if;
if counter = 500_000_00 then ADCout2 <= ADO; end if;
if counter = 800_000_00 then ADCout3 <= ADO; end if;

end process;
```

```
-------------------------------------------------------------------------------------
process(ADCout1, ADCout2, ADCout3) begin  -- Will Configure the Values Experimentally
--    if rising_edge(CLK100MHZ) and ADCout1 > "0000-0100-0000-0000" then
--        fako1 <= fako1 + 1;

--    if ADCout1 < "0010-0000-0000-0000"  then
--        enable <= '1'; else enable <= '0'; end if;

    if ADCout2 > "0100-1111-0111-0111" then
        control <= '1'; else control <= '0'; end if;

--    if ADCout3 < "0010-0000-0000-0000" then
--        WCL <= '1'; else WCL <= '0'; end if;

end process;
enable <= not(Realwaterfeed);

-------------------------------------------------------------
FoodContLow <= FoodContLevel;
--CC1 <= CleanCyc1;
WaterContLow <= not(Realwatercont);
--CC2 <= CleanCyc2;
Data1 <= ADCout1;
-------------------------------------------------------------
--process(CLK100MHZ) begin
--if rising_edge(CLK100MHZ) then
--    CC1 <= CleanCyc1;
--    CC2 <= CleanCyc2;
--end if;
--end process;


--process(FoodContLevel, WCL) begin
--    if falling_edge(FoodContLevel) then
--        if Cycles1 < 4 then
--            Cycles1 <= Cycles1+1;
```

```
--       else CleanCyc1 <= '1';
--       end if;
--    end if;
--    if falling_edge(WCL) then
--       if Cycles2 < 4 then
--          Cycles2 <= Cycles2 +1;
--       else CleanCyc2 <= '1';
--       end if;
--    end if;
--end process;
----------------------------------------------------------------
--process(RST1, RST2) begin
--if RST1 <= '1' then CleanCyc1 <= '0'; end if;
--if RST2 <= '1' then CleanCyc2 <= '0'; end if;
--end process;

-----------------------------------------
process(slow_clock) begin
if rising_edge(slow_clock) then
   if slow_counter = 200_000 or RST1 = '1' then
   slow_counter <= 0;
   else slow_counter <= slow_counter + 1;
   end if;
end if;
end process;

process(slow_clock) begin
if rising_edge(slow_clock) then
   if slow_counter2 = 200_000 or RST2 = '1' then
   slow_counter2 <= 0;
   else slow_counter2 <= slow_counter2 + 1;
   end if;
end if;
end process;

process(slow_counter) begin
if slow_counter > 172_800 then
   CC1 <= '1';
else CC1 <= '0';
end if;
end process;

process(slow_counter2) begin
if slow_counter2 > 86_400 then
```

```
    CC2 <= '1';
else CC2 <= '0';
end if;
end process;



end arc;
```

ADC_MAIN.vhd

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date: 12/15/2023 07:35:55 AM
-- Design Name:
-- Module Name: ADC_MAIN - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
--Dear Garda? Reader;
--When I wrote this code only god and
--I knew how it worked
--Now, only god knows it
--Therefore if you are trying to understand this code
--And by some miracle if you suceed, please also explain it to me

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;
```

```
-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;


--    input CLK100MHZ,
--    input vauxp6,
--    input vauxn6,
--    input vauxp7,
--    input vauxn7,
--    input vauxp15,
--    input vauxn15,
--    input vauxp14,
--    input vauxn14,
--    input vp_in,
--    input vn_in,
--    input [1:0] sw
entity ADC_MAIN is
Port (CLK100MHZ,vauxp6,vauxn6,vauxp7,vauxn7,vauxp15,vauxn15,vp_in,vn_in: in std_logic;
     Data1: out std_logic_vector(15 downto 0);
     sw: in std_logic_vector(1 downto 0)
       );
end ADC_MAIN;

architecture Behavioral of ADC_MAIN is
signal vauxp14,vauxn14: std_logic;
signal Da,databus7,databus15,databus14: std_logic_vector(15 downto 0);


component ADC
Port
(CLK100MHZ,vauxp6,vauxn6,vauxp7,vauxn7,vauxp15,vauxn15,vauxp14,vauxn14,vp_in,vn_in: in
std_logic;
     sw: in std_logic_vector(1 downto 0);
     Data1: out std_logic_vector(15 downto 0)
       );
end component;
begin

datat0: ADC port map (CLK100MHZ => CLK100MHZ,vauxp6
=>vauxp6,vauxn6=>vauxn6,vauxp7=>vauxp7,vauxn7=>vauxn7,vauxp15=>vauxp15,vauxn15=>va
uxn15,vauxp14=>vauxp14,vauxn14=>vauxn14,vp_in=>vp_in,vn_in=>vn_in,sw=>sw,Data1=>Data
1);
```

```
--datat1: ADC port map
(CLK100MHZ,vauxp6,vauxn6,vauxp7,vauxn7,vauxp15,vauxn15,vauxp14,vauxn14,vp_in,vn_in,"0
1",databus7);
--datat2: ADC port map
(CLK100MHZ,vauxp6,vauxn6,vauxp7,vauxn7,vauxp15,vauxn15,vauxp14,vauxn14,vp_in,vn_in,"1
0",databus15);



end Behavioral;
```

ADC.v (taken from Digilent Website)

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 02/12/2015 03:26:51 PM
// Design Name:
// Module Name: // Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Revision 0.02 - Fixed timing slack (ArtVVB 06/01/17)
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

// Taken from Xilinx website
module ADC(
   input CLK100MHZ,
   input vauxp6,
   input vauxn6,
   input vauxp7,
   input vauxn7,
   input vauxp15,
   input vauxn15,
   input vauxp14,
```

```
   input vauxn14,
   input vp_in,
   input vn_in,
   input [1:0] sw,
   output [15:0] data1
);

   wire enable;
   wire ready;
   wire [15:0] data;
   reg [6:0] Address_in;
         assign data1 = data;
//       //secen segment controller signals
//   reg [32:0] count;
//   localparam S_IDLE = 0;
//   localparam S_FRAME_WAIT = 1;
//   localparam S_CONVERSION = 2;
//   reg [1:0] state = S_IDLE;
//   reg [15:0] sseg_data;

//       //binary to decimal converter signals
//   reg b2d_start;
//   reg [15:0] b2d_din;
//   wire [15:0] b2d_dout;
//   wire b2d_done;

   //xadc instantiation connect the eoc_out .den_in to get continuous conversion
   xadc_wiz_0  XLXI_7 (
      .daddr_in(Address_in), //addresses can be found in the artix 7 XADC user guide DRP
register space
      .dclk_in(CLK100MHZ),
      .den_in(enable),
      .di_in(0),
      .dwe_in(0),
      .busy_out(),
      .vauxp6(vauxp6),
      .vauxn6(vauxn6),
      .vauxp7(vauxp7),
      .vauxn7(vauxn7),
      .vauxp14(vauxp14),
      .vauxn14(vauxn14),
      .vauxp15(vauxp15),
      .vauxn15(vauxn15),
      .vn_in(vn_in),
```

```
      .vp_in(vp_in),
      .alarm_out(),
      .do_out(data),
      //.reset_in(),
      .eoc_out(enable),
      .channel_out(),
      .drdy_out(ready)
   );

//   //led visual dmm
//   always @(posedge(CLK100MHZ)) begin
//      if(ready == 1'b1) begin
//         case (data[15:12])
//         1:  led <= 16'b11;
//         2:  led <= 16'b111;
//         3:  led <= 16'b1111;
//         4:  led <= 16'b11111;
//         5:  led <= 16'b111111;
//         6:  led <= 16'b1111111;
//         7:  led <= 16'b11111111;
//         8:  led <= 16'b111111111;
//         9:  led <= 16'b1111111111;
//         10: led <= 16'b11111111111;
//         11: led <= 16'b111111111111;
//         12: led <= 16'b1111111111111;
//         13: led <= 16'b11111111111111;
//         14: led <= 16'b111111111111111;
//         15: led <= 16'b1111111111111111;
//         default: led <= 16'b1;
//         endcase
//      end
//   end

   //binary to decimal conversion
//   always @ (posedge(CLK100MHZ)) begin
//      case (state)
//      S_IDLE: begin
//         state <= S_FRAME_WAIT;
//         count <= 'b0;
//      end
//      S_FRAME_WAIT: begin
//         if (count >= 10000000) begin
//            if (data > 16'hFFD0) begin
//               sseg_data <= 16'h1000;
```

```
//                state <= S_IDLE;
//             end else begin
//                b2d_start <= 1'b1;
//                b2d_din <= data;
//                state <= S_CONVERSION;
//             end
//          end else
//             count <= count + 1'b1;
//       end
//       S_CONVERSION: begin
//          b2d_start <= 1'b0;
//          if (b2d_done == 1'b1) begin
//             sseg_data <= b2d_dout;
//             state <= S_IDLE;
//          end
//       end
//       endcase
//    end

//    bin2dec m_b2d (
//       .clk(CLK100MHZ),
//       .start(b2d_start),
//       .din(b2d_din),
//       .done(b2d_done),
//       .dout(b2d_dout)
//    );

    always @(posedge(CLK100MHZ)) begin
       case(sw)
       0: Address_in <= 8'h16; // XA1/AD6
       1: Address_in <= 8'h1e; // XA2/AD14
       2: Address_in <= 8'h17; // XA3/AD7
       3: Address_in <= 8'h1f; // XA4/AD15
       endcase
    end

//    DigitToSeg segment1(
//       .in1(sseg_data[3:0]),
//       .in2(sseg_data[7:4]),
//       .in3(sseg_data[11:8]),
//       .in4(sseg_data[15:12]),
//       .in5(),
//       .in6(),
//       .in7(),
```

```
//       .in8(),
//       .mclk(CLK100MHZ),
//       .an(an),
//       .dp(dp),
//       .seg(seg)
//    );
endmodule
```

Servo.vhd

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date: 12/14/2023 11:36:19 PM
-- Design Name:
-- Module Name: servo - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;


entity servo is
 Port (control: in std_logic;
       CLK100MHZ: in std_logic;
       pwm: out std_logic
  );
end servo;

architecture Behavioral of servo is
signal pwm_s: std_logic;
```

```
signal inte: integer range 0 to 2_000_000 := 0;
begin
pwm <= pwm_s;
pwm_enable: process(CLK100MHZ) begin
if rising_edge(CLK100MHZ) then
if inte < 2_000_000 then
   inte <= inte + 1;
else
   inte <= 0;
end if;
if control = '0' then
   if inte < 100_000 then
      pwm_s <= '1';
   elsif inte < 1_999_999 and inte >= 100_000 then
      pwm_s <= '0';
   else
      pwm_s <= '0';
      inte <= 0;
   end if;
 else
      if inte < 150_000 then
      pwm_s <= '1';
   elsif inte < 1_999_999 and inte >= 150_000 then
      pwm_s <= '0';
   else
      pwm_s <= '0';
   end if;
end if;
end if;
end process;
end Behavioral;
```

Waterpump.vhd

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date: 12/14/2023 11:36:19 PM
-- Design Name:
-- Module Name: servo - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
```

```
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;


entity servo is
 Port (control: in std_logic;
       CLK100MHZ: in std_logic;
       pwm: out std_logic
  );
end servo;

architecture Behavioral of servo is
signal pwm_s: std_logic;
signal inte: integer range 0 to 2_000_000 := 0;
begin
pwm <= pwm_s;
pwm_enable: process(CLK100MHZ) begin
if rising_edge(CLK100MHZ) then
if inte < 2_000_000 then
   inte <= inte + 1;
else
   inte <= 0;
end if;
if control = '0' then
   if inte < 100_000 then
      pwm_s <= '1';
   elsif inte < 1_999_999 and inte >= 100_000 then
      pwm_s <= '0';
   else
      pwm_s <= '0';
      inte <= 0;
   end if;
 else
```

```
      if inte < 150_000 then
      pwm_s <= '1';
   elsif inte < 1_999_999 and inte >= 150_000 then
      pwm_s <= '0';
   else
      pwm_s <= '0';
   end if;
end if;
end if;
end process;
end Behavioral;
```

xadc_wiz_0.vhd (Created with XADC wizard)

```
-- file: xadc_wiz_0.vhd
-- (c) Copyright 2009 - 2013 Xilinx, Inc. All rights reserved.
--
-- This file contains confidential and proprietary information
-- of Xilinx, Inc. and is protected under U.S. and
-- international copyright and other intellectual property
-- laws.
--
-- DISCLAIMER
-- This disclaimer is not a license and does not grant any
-- rights to the materials distributed herewith. Except as
-- otherwise provided in a valid license issued to you by
-- Xilinx, and to the maximum extent permitted by applicable
-- law: (1) THESE MATERIALS ARE MADE AVAILABLE "AS IS" AND
-- WITH ALL FAULTS, AND XILINX HEREBY DISCLAIMS ALL WARRANTIES
-- AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING
-- BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-
-- INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and
-- (2) Xilinx shall not be liable (whether in contract or tort,
-- including negligence, or under any other theory of
-- liability) for any loss or damage of any kind or nature
-- related to, arising under or in connection with these
-- materials, including for any direct, or any indirect,
-- special, incidental, or consequential loss or damage
-- (including loss of data, profits, goodwill, or any type of
-- loss or damage suffered as a result of any action brought
-- by a third party) even if such damage or loss was
-- reasonably foreseeable or Xilinx had been advised of the
-- possibility of the same.
--
```

```
-- CRITICAL APPLICATIONS
-- Xilinx products are not designed or intended to be fail-
-- safe, or for use in any application requiring fail-safe
-- performance, such as life-support or safety devices or
-- systems, Class III medical devices, nuclear facilities,
-- applications related to the deployment of airbags, or any
-- other applications that could lead to death, personal
-- injury, or severe property or environmental damage
-- (individually and collectively, "Critical
-- Applications"). Customer assumes the sole risk and
-- liability of any use of Xilinx products in Critical
-- Applications, subject only to applicable laws and
-- regulations governing limitations on product liability.
--
-- THIS COPYRIGHT NOTICE AND DISCLAIMER MUST BE RETAINED AS
-- PART OF THIS FILE AT ALL TIMES.

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
Library UNISIM;
use UNISIM.VCOMPONENTS.ALL;

entity xadc_wiz_0 is
  port
  (
   daddr_in       : in  STD_LOGIC_VECTOR (6 downto 0);     -- Address bus for the dynamic
reconfiguration port
   den_in         : in  STD_LOGIC;                    -- Enable Signal for the dynamic reconfiguration
port
   di_in          : in  STD_LOGIC_VECTOR (15 downto 0);    -- Input data bus for the dynamic
reconfiguration port
   dwe_in         : in  STD_LOGIC;                     -- Write Enable for the dynamic reconfiguration
port
   do_out         : out  STD_LOGIC_VECTOR (15 downto 0);   -- Output data bus for dynamic
reconfiguration port
   drdy_out       : out  STD_LOGIC;                      -- Data ready signal for the dynamic
reconfiguration port
   dclk_in        : in  STD_LOGIC;                    -- Clock input for the dynamic reconfiguration port
   vauxp6         : in  STD_LOGIC;                     -- Auxiliary Channel 6
   vauxn6         : in  STD_LOGIC;
   vauxp7         : in  STD_LOGIC;                     -- Auxiliary Channel 7
   vauxn7         : in  STD_LOGIC;
   vauxp14        : in  STD_LOGIC;                      -- Auxiliary Channel 14
```

```
   vauxn14        : in  STD_LOGIC;
   vauxp15        : in  STD_LOGIC;                    -- Auxiliary Channel 15
   vauxn15        : in  STD_LOGIC;
   busy_out       : out  STD_LOGIC;                    -- ADC Busy signal
   channel_out    : out  STD_LOGIC_VECTOR (4 downto 0);    -- Channel Selection Outputs
   eoc_out        : out  STD_LOGIC;                    -- End of Conversion Signal
   eos_out        : out  STD_LOGIC;                    -- End of Sequence Signal
   alarm_out      : out STD_LOGIC;                     -- OR'ed output of all the Alarms
   vp_in          : in  STD_LOGIC;                   -- Dedicated Analog Input Pair
   vn_in          : in  STD_LOGIC
);
end xadc_wiz_0;

architecture xilinx of xadc_wiz_0 is

  attribute CORE_GENERATION_INFO : string;
  attribute CORE_GENERATION_INFO of xilinx : architecture is
"xadc_wiz_0,xadc_wiz_v3_3_8,{component_name=xadc_wiz_0,enable_axi=false,enable_axi4str
eam=false,dclk_frequency=100,enable_busy=true,enable_convst=false,enable_convstclk=false,
enable_dclk=true,enable_drp=true,enable_eoc=true,enable_eos=true,enable_vbram_alaram=f
alse,enable_vccddro_alaram=false,enable_Vccint_Alaram=false,enable_Vccaux_alaram=falseen
able_vccpaux_alaram=false,enable_vccpint_alaram=false,ot_alaram=false,user_temp_alaram=f
alse,timing_mode=continuous,channel_averaging=None,sequencer_mode=on,startup_channel
_selection=contineous_sequence}";


  signal FLOAT_VCCAUX_ALARM : std_logic;
  signal FLOAT_VCCINT_ALARM : std_logic;
  signal FLOAT_USER_TEMP_ALARM : std_logic;
  signal FLOAT_VBRAM_ALARM : std_logic;
  signal FLOAT_MUXADDR : std_logic_vector (4 downto 0);
  signal aux_channel_p : std_logic_vector (15 downto 0);
  signal aux_channel_n : std_logic_vector (15 downto 0);
  signal alm_int : std_logic_vector (7 downto 0);

begin

      alarm_out <= alm_int(7);

       aux_channel_p(0) <= '0';
       aux_channel_n(0) <= '0';

       aux_channel_p(1) <= '0';
       aux_channel_n(1) <= '0';
```

```
    aux_channel_p(2) <= '0';
    aux_channel_n(2) <= '0';

    aux_channel_p(3) <= '0';
    aux_channel_n(3) <= '0';

    aux_channel_p(4) <= '0';
    aux_channel_n(4) <= '0';

    aux_channel_p(5) <= '0';
    aux_channel_n(5) <= '0';

    aux_channel_p(6) <= vauxp6;
    aux_channel_n(6) <= vauxn6;

    aux_channel_p(7) <= vauxp7;
    aux_channel_n(7) <= vauxn7;

    aux_channel_p(8) <= '0';
    aux_channel_n(8) <= '0';

    aux_channel_p(9) <= '0';
    aux_channel_n(9) <= '0';

    aux_channel_p(10) <= '0';
    aux_channel_n(10) <= '0';

    aux_channel_p(11) <= '0';
    aux_channel_n(11) <= '0';

    aux_channel_p(12) <= '0';
    aux_channel_n(12) <= '0';

    aux_channel_p(13) <= '0';
    aux_channel_n(13) <= '0';

    aux_channel_p(14) <= vauxp14;
    aux_channel_n(14) <= vauxn14;

    aux_channel_p(15) <= vauxp15;
    aux_channel_n(15) <= vauxn15;

U0 : XADC
```

```
    generic map(
      INIT_40 => X"0000", -- config reg 0
      INIT_41 => X"21AF", -- config reg 1
      INIT_42 => X"0400", -- config reg 2
      INIT_48 => X"0000", -- Sequencer channel selection
      INIT_49 => X"C0C0", -- Sequencer channel selection
      INIT_4A => X"0000", -- Sequencer Average selection
      INIT_4B => X"0000", -- Sequencer Average selection
      INIT_4C => X"0000", -- Sequencer Bipolar selection
      INIT_4D => X"0000", -- Sequencer Bipolar selection
      INIT_4E => X"0000", -- Sequencer Acq time selection
      INIT_4F => X"0000", -- Sequencer Acq time selection
      INIT_50 => X"B5ED", -- Temp alarm trigger
      INIT_51 => X"57E4", -- Vccint upper alarm limit
      INIT_52 => X"A147", -- Vccaux upper alarm limit
      INIT_53 => X"CA33",  -- Temp alarm OT upper
      INIT_54 => X"A93A", -- Temp alarm reset
      INIT_55 => X"52C6", -- Vccint lower alarm limit
      INIT_56 => X"9555", -- Vccaux lower alarm limit
      INIT_57 => X"AE4E",  -- Temp alarm OT reset
      INIT_58 => X"5999",  -- Vccbram upper alarm limit
      INIT_5C => X"5111",  -- Vccbram lower alarm limit
      SIM_DEVICE => "7SERIES",
      SIM_MONITOR_FILE => "design.txt"
      )

port map (
      CONVST            => '0',
      CONVSTCLK         => '0',
      DADDR(6 downto 0)   => daddr_in(6 downto 0),
      DCLK              => dclk_in,
      DEN               => den_in,
      DI(15 downto 0)     => di_in(15 downto 0),
      DWE               => dwe_in,
      RESET             => '0',
      VAUXN(15 downto 0)  => aux_channel_n(15 downto 0),
      VAUXP(15 downto 0)  => aux_channel_p(15 downto 0),
      ALM               => alm_int,
      BUSY              => busy_out,
      CHANNEL(4 downto 0) => channel_out(4 downto 0),
      DO(15 downto 0)     => do_out(15 downto 0),
      DRDY              => drdy_out,
      EOC               => eoc_out,
      EOS               => eos_out,
```

```
      JTAGBUSY          => open,
      JTAGLOCKED        => open,
      JTAGMODIFIED      => open,
      OT                => open,

      MUXADDR           => FLOAT_MUXADDR,
      VN                => vn_in,
      VP                => vp_in
       );
end xilinx;
```