FPGA Synthesis

1 Purpose

This laboratory focuses on the design and implementation flow of a fully working digital system targeting an FPGA development board. The purpose of this laboratory work is to show how a FPGA device can be programmed.

2 Introduction

In order to design digital systems using programmable logic devices such as FPGAs, computer aided design (CAD) software products are used. These software solutions offer the whole package, from the early designing stage to the final FPGA implementation stage. Thus, 5 steps can be distinguished:

1. **Description:** VDHL source code
2. **Synthesis:** compile VHDL code, transform description into a netlist (basic logic gates and interconnections)
3. **Functional simulation:** simulation based on test benches and waveforms (usually from the IDE, such as Vivado Simulator, Xilinx ISE Simulator, ModelSim etc.)
4. **Implementation:** adapting the netlist to the available resources of the device (technology mapping, place and route)
5. **Configuration:** programming the device

Each stage mentioned above is important in order to design a fully working digital system, on a FPGA development board. The examples in this laboratory work are customized for the Basys 3 FPGA board, but can be easily changed to match the programming requirements of other development boards.

3 FPGA Syntesis of a FIFO Memory

This section presents a design example of a FIFO memory that is based on the VHDL language and uses the Vivado environment for both simulation and implementation.

3.1 FIFO Memory Design Considerations

Figure 1 below illustrates the block diagram of the FIFO memory. The capacity of the designed memory is of 8 words of 8 bits each.

When the read signal rd is asserted, the output `data_out` of the memory must be enabled. When the read signal `rd` is not asserted, the output must be placed in the high-impedance state (`"ZZZZZZZZ"`). When the write signal `wr` is asserted, the value from the `data_in` input of the memory must be written into one of the 8 registers. The read (`rdptr`) and write (`wrptr`) pointers indicate which register to read and which register to write. To increment the read pointer, the `rdinc` signal is used, and to increment the write pointer, the `wrinc` signal is used.

The design also needs a control unit, some debounce circuits that ensure that the input received through button inputs is correctly interpreted and a 7-segments display module. The block design of this top level design is visible in Figure 2.

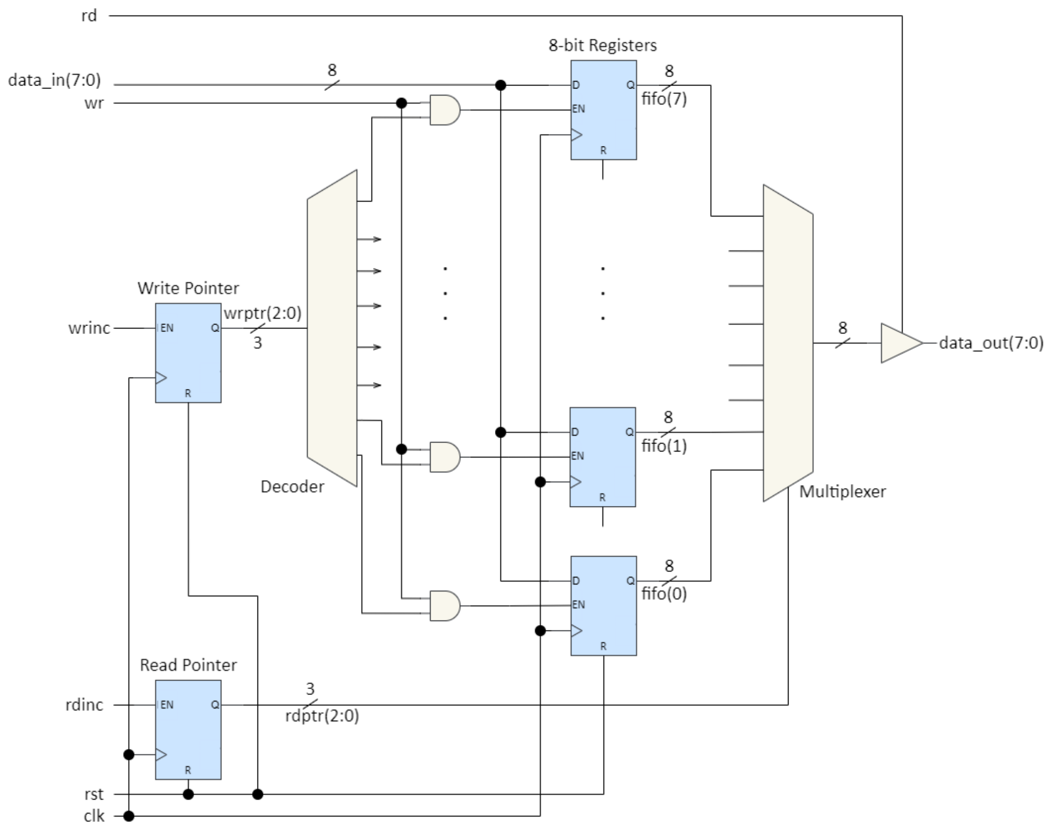

Figure 1: Block design of the FIFO memory

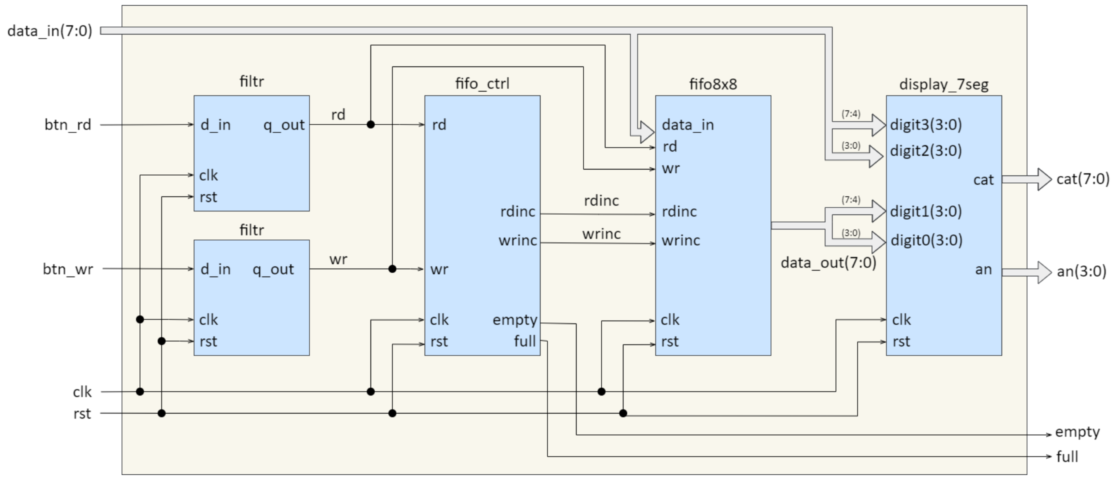

Figure 2: Top level block design targeting the Basys 3 device

The control unit has as main inputs the read and write signals of the memory (`rd` and `wr`), which generate the `rdinc` and `wrinc` signals for incrementing the read and write pointers from the FIFO memory module.

Also, the control unit will generate the `empty` and `full` status signals (optional):

- `empty` signal is set to logical 1 when no words have been written in the FIFO memory or all words have been read
- `full` signal is set to logical 1 when the FIFO memory contains 8 words that have not been read

The `filtr` block in Figure 2 is a debouncing module for the buttons used to generate the `rd` and `wr` signals. When one of these buttons is pressed, the corresponding pin of the FPGA device will be connected to logical 1. This module will be instantiated 2 times in the top-level module and will generate a single pulse when the corresponding button is pressed.

3.2 Basys 3 Implementation Guide

This sub-section aims to provide a guideline on the development and implementation steps needed to design the FIFO memory and to use it to program a Basys 3 FPGA development board.

Step 1. Create a new RTL project using Vivado IDE targeting the Basys 3 device

Open Vivado, click on *File* → *Project* → *New...* and in the opened window click *Next*. Provide the path or browse to the directory where you want to create the project and click *Next*. In the *Project Type* window select *RTL Project* and tick the *Do not specify sources at this time* option. The device properties that you need to choose are:

- **Product category:** All
- **Family:** Artix-7
- **Package:** cpg236
- **Speed grade:** -1

Then select from the table the **xc7a35tcpg236-1** board and click *Next*. Click *Finish*.

Step 2. Implement and simulate the behavior of the FIFO Control Unit

In the *Sources* tab, right-click on *Design Sources* and then click on *Add Sources....* In the newly opened window select *Add or create design sources* and click on *Next* button. Then select *Create File*, type `fifo_control` in the *File name* text field, make sure that the selected *File type* is VHDL and press *OK*. Then, click on *Finish* button. In the *Define Module* window specify the input and output ports of the control unit according to Figure ... then press *OK*. Alternatively, you can leave the ports undefined for the moment and manually define them in the created `.vhd` file.

Write the code of the architecture of the `fifo_control` entity in `fifo_control.vhd` file. For simplicity, you can follow the behavioral programming model.

After implementing the FIFO Control Unit according to the design considerations in Sub-section 3.1, simulate the behavior of the implemented control unit. Create a testbench file to easily simulate the implemented design. In the *Sources* tab, right-click on *Simulation Sources* and then click on *Add Sources....* In the newly opened window select *Add or create simulation sources* and click on *Next* button. Then select *Create File*, type `Tb_fifo_control` in the *File name* text field, make sure that the selected *File type* is VHDL and press *OK*. Then, click on *Finish* button. In the *Define Module* window leave the input and output ports of the entity undefined as a testbench simulation entity does not have any inputs and outputs.

In the created testbench file declare the `fifo_control` entity as a component and its inputs and outputs as internal signals. Then, instantiate the `fifo_control` and map its ports to the previously declared signals. Assign values to the signals corresponding to the inputs of the `fifo_control` unit and simulate the behaviour using the Vivado integrated simulator. For this, right-click on the `Tb_fifo_control.vhd` file in *Simulation Sources* → *sim1* within the *Sources* tab and select *Set as Top*. In the *Simulation* tab within the *Flow Manager* click on *Run Behavioral Simulation* to start the integrated simulator. Simulate the design to make sure that the output provided by the `fifo_control` unit is the expected one for every inputs combination.

Step 3. Implement and simulate the behavior of the FIFO Memory

Follow the same flow as in **Step 2** to implement and simulate the FIFO memory, according to the design specifications in Sub-section 3.1.

To implement the FIFO module, write a process for each of the following elements of the memory:

- Read pointer
- Write pointer
- Decoder
- Register set
- Multiplexer
- Tri-state buffer

Alternatively, you can follow a behavioral programming model.

Step 4. Include the debouncing circuit

In the *Sources* tab, right-click on *Design Sources* and then click on *Add Sources...*. Select *Add or create design sources* and press *Next*. Click on *Add files*. In the navigation window select for the *Files of type* field the value *All Files* and navigate to the location of the `debouncer.vhd` file on your computer. After finding the file, select it, press *OK* and then *Finish*.

The content of the `debouncer.vhd` file is also listed in Section A.1 of the Code Annex A.

Alternatively, you can create yourself a debouncer module using the *Language Templates* within the Vivado editor. Navigate to *Tools* → *Language Templates*. In the newly opened window, expand **VHDL** → *Synthesis Constructs* → *Coding Examples* → *Misc* → *Debouncer Circuit*. Copy the code and paste it into a newly created `.vhd` file. Name the file `debouncer.vhd`.

Step 5. Include the 7-segment display module

Follow the instructions provided in **Step 4** to import the 7-segments display in file `display_7seg.vhd`. The content of the file is also listed in Section A.2 of the Code Annex A.

You can also create yourself a 7-segment display using *Language Templates*. To do this, go to *Tools* → *Language Templates*. In the newly opened window, expand **VHDL** → *Synthesis Constructs* → *Coding Examples* → *Misc* → *7-Segment Display Hex Conversion*. Copy the code and use it to create a 7-segment display and include it in your project.

Step 5. Create the top level design targeting the Basys 3 device

In order to implement the FIFO memory on the Basys 3 development board, you must first define the peripherals of the board which will be used:

- Input data will be entered from the `SW7-SW0` slide switches
- The status of the slide switches will be displayed on the 2 left-hand side digits of the 7 segments display
- The byte read from the memory will be displayed on the 2 right-hand side digits of the 7 segments display
- `rst` signal assigned to button left (`BTNL`)
- `rd` signal assigned to button up (`BTNU`)
- `wr` signal assigned to button down (`BTND`)
- `full` status signal will be displayed on `LD0` Led
- `empty` status signal will be displayed on `LD1` Led

The Input/Output (I/O) devices on the Basys 3 board and their corresponding sites are depicted in Figure 3.

To implement the final design, create the top level entity of the design. Create a design source file and name it `basys3_fifo8x8.vhd`, following the instructions in **Step 2**. The ports of the entity in this file should mach the I/O devices of the Basys 3 development board, shown in Figure 3.

The structure of the entity should be the following:

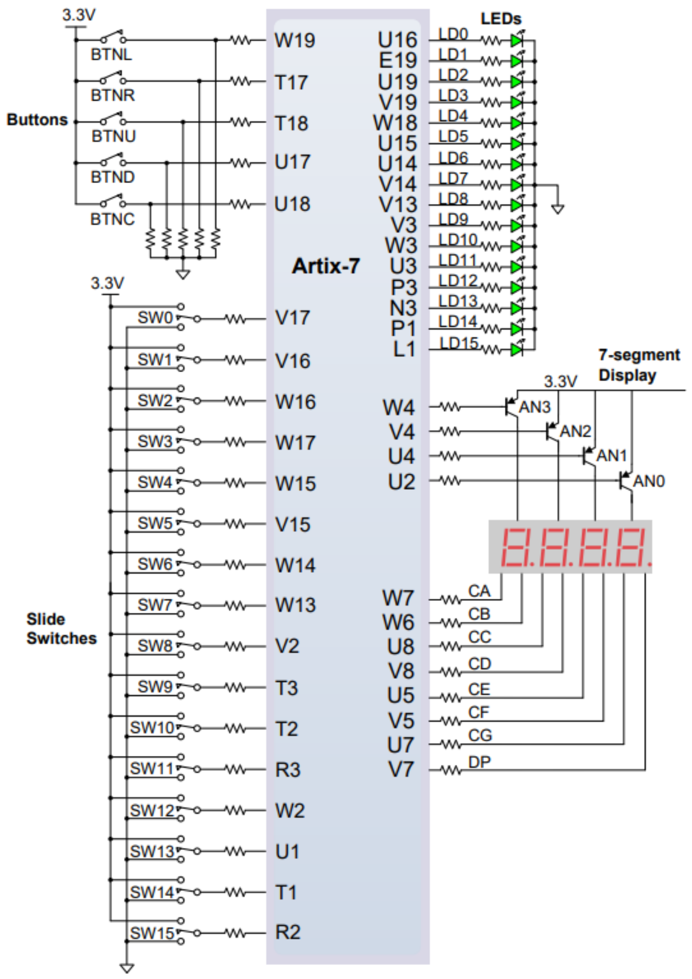

Figure 3: General purpose I/O devices on the Basys 3

```
entity basys3_fifo8x8 is
    Port ( clk : in STD_LOGIC;
           btn : in STD_LOGIC_VECTOR (4 downto 0);
           sw : in STD_LOGIC_VECTOR (15 downto 0);
           led : out STD_LOGIC_VECTOR (15 downto 0);
           an : out STD_LOGIC_VECTOR (3 downto 0);
           cat : out STD_LOGIC_VECTOR (6 downto 0));
end basys3_fifo8x8;
```

Within the architecture of the `basys3_fifo8x8` entity, declare the `debouncer`, `fifo_control`, `fifo8x8` and `display_7seg` as components in the declarative part. Then, instantiate the components and link them using internal signal to create the final design shown in Figure ... .

Note. To follow the previously stated mapping between the signals and the buttons and leds on the board, you should use the `btn` input port and the `led` output port as follows:

Signal	Port	Site
`rst`	`btn(2)`	W19
`rd`	`btn(1)`	T18
`wr`	`btn(4)`	U17
`full`	`led(0)`	U16
`empty`	`led(1)`	E19

Table 1: The mapping between input signals and top entity button input ports

After implementing the top level entity, right-click on the `basys3_fifo8x8.vhd` file within the *Design Sources* section of the *Sources* tab and select *Set as Top*.

Step 6. Import the Basys 3 `.xdc` constraints file

In the *Sources* tab, right-click on *Constraints* and then click on *Add Sources...*. Select *Add or create constraints* and press *Next*. Click on *Add files*. Navigate to the location of the `basys3_constr.xdc` file on your computer. After finding the file, select it, press *OK*. Check that *Copy constraints files into project* is checked and then press *Finish*. Roght-click on the file in *Constraints* section of the *Sources* tab and select *Set as Target Constraint File*.

The content of the `basys3_constr.xdc` file is also listed in Section A.3 of the Code Apendix A.

Step 7. Run Synthesis and Implementation

In the *Synthesis* menu of the *Flow Navigator* click on *Run Synthesis* and in the newly opened window press *OK*. When the *Synthesis Completed* window appears, click on *Cancel*.

Similarly, in the *Implementation* menu of the *Flow Navigator* click on *Run Implementation* and in the newly opened window press *OK*. When the *Implementation Completed* window appears, also click on *Cancel*.

Step 8. Generate the bitstream file

In the *Program and Debug* menu of the *Flow Navigator* click on *Generate Bitstream* and press *OK* in the dialog. When the *Bitstream Generation Completed* window appears, click on *Cancel*. A `.bit` file will be created. This file will be uploaded on the board in the next step.

Step 8. Upload the `.bit` programming file on the Basys 3 board

Connect the Basys 3 development board to the computer using the USB port and make sure that the POWER switch is set to ON. Go to *Flow Navigator* → *Open Hardware Manager* → *Open Target* → *Auto Connect*.

If the board was recognized by the computer, the *Program Device* option in *Open Hardware Manager* should be available. If not, go to *Open Hardware Manager* → *Open Target* → *Open New Target...* and follow the steps by pressing the *Next* button in the dialog.

When the *Program Device* option is available, click on it, select the **xc7a357_0** device and in the opened window navigate to the `.bit` file, located in the `<project_name>.runs/impl1` sub-directory of your project. Select the `.bit` file and click on *Program*. The *.bit* file is now uploaded on the board and you can use the switches and buttons on the board to test your design.

4 Exercise

Follow the steps in Sub-section 3.2 to implement the FIFO memory and use the design to program a Basys 3 development board.

References

[1] Digilent Basys 3™ FPGA Board Reference Manual [accessed Oct. 2024], `https://digilent.com/reference/_media/reference/programmable-logic/basys-3/basys3_rm.pdf?srsltid=AfmBOoohlun6PZGSOibCo6xLfaa2oMmA_BoNwQ-lY3hdSjRcfziJ_ABl`

[2] What is FIFO? Synchronous FIFO, Asynchronous FIFO [accessed Oct. 2024], `https://semiconductorclub.com/what-is-fifo-synchronous-fifo-asynchronous-fifo/`

[3] VHDL and FPGA terminology - Synthesis [accessed Oct. 2024], `https://vhdlwhiz.com/terminology/synthesis/`

A Code appendix

A.1 Debouncer Circuit

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

entity debouncer is
  Port ( clk : in std_logic;
        btn : in std_logic;
        en : out std_logic );
end debouncer;

architecture Behavioral of debouncer is

signal count : std_logic_vector(15 downto 0) := (others => '0');
signal q1 : std_logic := '0';
signal q2 : std_logic := '0';
signal q3 : std_logic := '0';

begin
    process(clk)
    begin
        if rising_edge(clk) then
            count <= count + 1;
        end if;
    end process;

    process(clk)
    begin
        if rising_edge(clk) then
            if count(15 downto 0) = x"1111" then
                q1 <= btn;
            end if;
        end if;
    end process;

    process(clk)
    begin
        if rising_edge(clk) then
            q2 <= q1;
            q3 <= q2;
        end if;
    end process;

    en <= q2 and (not q3);
end Behavioral;
```

A.2 7-Segment Display

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

entity display_7seg is
    Port ( digit0 : in STD_LOGIC_VECTOR (3 downto 0);
           digit1 : in STD_LOGIC_VECTOR (3 downto 0);
           digit2 : in STD_LOGIC_VECTOR (3 downto 0);
           digit3 : in STD_LOGIC_VECTOR (3 downto 0);
           clk : in STD_LOGIC;
           cat : out STD_LOGIC_VECTOR (6 downto 0);
           an : out STD_LOGIC_VECTOR (3 downto 0));
end display_7seg;

architecture Behavioral of display_7seg is

signal cnt : STD_LOGIC_VECTOR (15 downto 0) := (others => '0');
```

```vhdl
signal digit_to_display : STD_LOGIC_VECTOR (3 downto 0) := (others => '0');

begin
    process (clk)
    begin
        if rising_edge(clk) then
            cnt <= cnt + 1;
        end if;
    end process;

    digit_to_display <= digit0 when cnt(15 downto 14) = "00" else
                        digit1 when cnt(15 downto 14) = "01" else
                        digit2 when cnt(15 downto 14) = "10" else
                        digit3;

    an <= "1110" when cnt(15 downto 14) = "00" else
          "1101" when cnt(15 downto 14) = "01" else
          "1011" when cnt(15 downto 14) = "10" else
          "0111";

    -- HEX-to-seven-segment decoder
    --   HEX:   in    STD_LOGIC_VECTOR (3 downto 0);
    --   LED:   out   STD_LOGIC_VECTOR (6 downto 0);
    --
    -- segment encoinputg
    --      0
    --     ---
    --  5 |   | 1
    --     ---   <- 6
    --  4 |   | 2
    --     ---
    --      3

    with digit_to_display select
         cat <= "1111001" when "0001",   --1
                "0100100" when "0010",   --2
                "0110000" when "0011",   --3
                "0011001" when "0100",   --4
                "0010010" when "0101",   --5
                "0000010" when "0110",   --6
                "1111000" when "0111",   --7
                "0000000" when "1000",   --8
                "0010000" when "1001",   --9
                "0001000" when "1010",   --A
                "0000011" when "1011",   --b
                "1000110" when "1100",   --C
                "0100001" when "1101",   --d
                "0000110" when "1110",   --E
                "0001110" when "1111",   --F
                "1000000" when others;   --0

end Behavioral;
```

A.3 Basys 3 .xdc constraints file

```
# This file is a general .xdc for the Basys3 rev B board
# To use it in a project:
# - uncomment the lines corresponding to used pins
# - rename the used ports (in each line, after get_ports) according to the top level signal
# names in the project

# Clock signal
set_property PACKAGE_PIN W5 [get_ports clk]
 set_property IOSTANDARD LVCMOS33 [get_ports clk]
 #create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk]

# Switches
```

```
set_property PACKAGE_PIN V17 [get_ports {sw[0]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {sw[0]}]
set_property PACKAGE_PIN V16 [get_ports {sw[1]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {sw[1]}]
set_property PACKAGE_PIN W16 [get_ports {sw[2]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {sw[2]}]
set_property PACKAGE_PIN W17 [get_ports {sw[3]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {sw[3]}]
set_property PACKAGE_PIN W15 [get_ports {sw[4]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {sw[4]}]
set_property PACKAGE_PIN V15 [get_ports {sw[5]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {sw[5]}]
set_property PACKAGE_PIN W14 [get_ports {sw[6]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {sw[6]}]
set_property PACKAGE_PIN W13 [get_ports {sw[7]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {sw[7]}]
set_property PACKAGE_PIN V2 [get_ports {sw[8]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {sw[8]}]
set_property PACKAGE_PIN T3 [get_ports {sw[9]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {sw[9]}]
set_property PACKAGE_PIN T2 [get_ports {sw[10]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {sw[10]}]
set_property PACKAGE_PIN R3 [get_ports {sw[11]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {sw[11]}]
set_property PACKAGE_PIN W2 [get_ports {sw[12]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {sw[12]}]
set_property PACKAGE_PIN U1 [get_ports {sw[13]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {sw[13]}]
set_property PACKAGE_PIN T1 [get_ports {sw[14]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {sw[14]}]
set_property PACKAGE_PIN R2 [get_ports {sw[15]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {sw[15]}]


# LEDs
set_property PACKAGE_PIN U16 [get_ports {led[0]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {led[0]}]
set_property PACKAGE_PIN E19 [get_ports {led[1]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {led[1]}]
set_property PACKAGE_PIN U19 [get_ports {led[2]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {led[2]}]
set_property PACKAGE_PIN V19 [get_ports {led[3]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {led[3]}]
set_property PACKAGE_PIN W18 [get_ports {led[4]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {led[4]}]
set_property PACKAGE_PIN U15 [get_ports {led[5]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {led[5]}]
set_property PACKAGE_PIN U14 [get_ports {led[6]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {led[6]}]
set_property PACKAGE_PIN V14 [get_ports {led[7]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {led[7]}]
set_property PACKAGE_PIN V13 [get_ports {led[8]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {led[8]}]
set_property PACKAGE_PIN V3 [get_ports {led[9]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {led[9]}]
set_property PACKAGE_PIN W3 [get_ports {led[10]}]
```

```
 set_property IOSTANDARD LVCMOS33 [get_ports {led[10]}]
set_property PACKAGE_PIN U3 [get_ports {led[11]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {led[11]}]
set_property PACKAGE_PIN P3 [get_ports {led[12]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {led[12]}]
set_property PACKAGE_PIN N3 [get_ports {led[13]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {led[13]}]
set_property PACKAGE_PIN P1 [get_ports {led[14]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {led[14]}]
set_property PACKAGE_PIN L1 [get_ports {led[15]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {led[15]}]


#7 catment display
set_property PACKAGE_PIN W7 [get_ports {cat[0]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {cat[0]}]
set_property PACKAGE_PIN W6 [get_ports {cat[1]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {cat[1]}]
set_property PACKAGE_PIN U8 [get_ports {cat[2]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {cat[2]}]
set_property PACKAGE_PIN V8 [get_ports {cat[3]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {cat[3]}]
set_property PACKAGE_PIN U5 [get_ports {cat[4]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {cat[4]}]
set_property PACKAGE_PIN V5 [get_ports {cat[5]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {cat[5]}]
set_property PACKAGE_PIN U7 [get_ports {cat[6]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {cat[6]}]



set_property PACKAGE_PIN U2 [get_ports {an[0]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {an[0]}]
set_property PACKAGE_PIN U4 [get_ports {an[1]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {an[1]}]
set_property PACKAGE_PIN V4 [get_ports {an[2]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {an[2]}]
set_property PACKAGE_PIN W4 [get_ports {an[3]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {an[3]}]


#Buttons
set_property PACKAGE_PIN U18 [get_ports {btn[0]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {btn[0]}]
set_property PACKAGE_PIN T18 [get_ports {btn[1]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {btn[1]}]
set_property PACKAGE_PIN W19 [get_ports {btn[2]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {btn[2]}]
set_property PACKAGE_PIN T17 [get_ports btn[3]]
 set_property IOSTANDARD LVCMOS33 [get_ports {btn[3]}]
set_property PACKAGE_PIN U17 [get_ports {btn[4]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {btn[4]}]
```