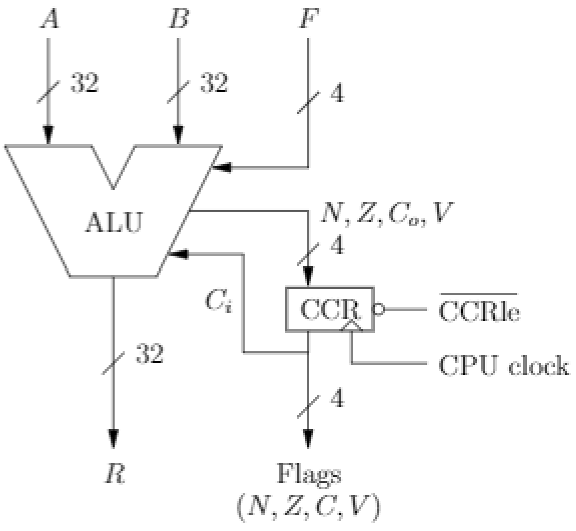
Lab 5 Report

ALU

CENG 342 Digital Systems



Samuel Donovan

Samantha Pfeiffer

2021-02-16

OVERVIEW

The purpose of this lab was to build an ALU (Arithmetic Logic Unit) that combined our Adder/Subtractor and Barrel Shifter, designed in previous labs, with a Logic Unit into one module.

--- ALU ---

An ALU (Arithmetic Logic Unit) is responsible for the mathematical and logical operations within a CPU.

While some ALUs may contain complex circuits for things such as multiplication and trigonometric functions, the most basic ALU consists of an Adder/Subtractor (responsible for additions and subtractions), a Logic Unit (responsible for simple logical operations such as AND/OR/NOT), and a shifter (responsible for logical and arithmetic shift operations).

Every function has one or two inputs, ***A*** and ***B***, and an output, ***R***. Which operation is performed is dependent on the input ***F***.

In addition to performing these operations on one or two inputs and providing the result, the ALU also commonly sets or clears flags in CPSR (current program status register), also known as a CCR (condition code register).

While the CPSR contains numerous status flags, the ALU is only concerned with 4: ***N***egative, ***Z***ero , ***C***arry, and o***V***erflow. As not all flags are valid for every operation (e.g. AND will not have a carry), the status bits are cleared (set to 0) before each operation.

When an unsupported ***F*** code is given, ***R*** is set to all 0s, as are ***N***, ***Z***, ***C***, and ***V***.

Table 1 shows the valid ***F*** codes, as well as their defined operations and which status bits can be set as a result of the operation.

Table 1 ALU Operations

F	Op	Description	R	Status Bits
0000	ADD	Add	$A + B$	N, Z, C_o, V
0010	ADC	Add w/ Carry	$A + B + C_i$	N, Z, C_o, V
0100	SUB	Subtract	$A + \overline{B} + 1$	N, Z, C_o, V
0110	SBC	Subtract w/ Carry	$A + \overline{B} + C_i$	N, Z, C_o, V
1001	NOT	Bitwise NOT	$\overline{B}$	N, Z
1011	AND	Bitwise AND	$A \wedge B$	N, Z
1101	OR	Bitwise OR	$A \vee B$	N, Z
1111	XOR	Bitwise XOR	$A \oplus B$	N, Z
1000	A	Pass A	A	N, Z
1010	LSL	Logical Shift Left	$A << B$	N, Z, C_o, V
1100	LSR	Logical Shift Right	$A >> B$	N, Z, C_o
1110	ASR	Arithmetic Shift Right	$A >> B$(signed)	N, Z, C_o

DESIGN

All designs were made for a Nexys A7-100T (xc7a100tcsg324-1) FPGA board in VHDL using Vivado 2020.2.

--- LOGIC UNIT ---

The final functional block needed for the ALU consisted of four bitwise logic operations, referred to as the Logic Unit. The VHDL code for this block can be found in **Appendix B**. A `with select` statement is used to conditionally set a signal ***R*** based on the value of the input ***FUNC***. For each value of ***FUNC***, different bitwise logic is performed on inputs ***A*** and ***B***. The operations are listed in **Table 1**, and the corresponding ***FUNC*** values can be derived by looking at the center 2 bits ***F***. After ***R*** is set, it is assigned to the output ***Y***. ***R*** is used as an intermediary signal so that the ***N*** and ***Z*** flags can be set accordingly.

--- ALU ---

The ALU was designed using the adder/subtractor and barrel shifter from **Lab 4**, as well as the logic unit described above. As each of these units was tested and verified function prior to use in the ALU, combining them into a single unit was of relative ease. The VHDL code for the ALU is in **Appendix A**.

It was noticed that the input ***F***, which defines the function as seen in **Table 1**, followed the pattern that the first and last bits determined which sub-unit was used, and the two inner bits determined which operation was being performed within the units. Therefore, in the VHDL code ***F*** is divided into `FUNC_SEL` and `BLOCK_SEL` with the respective bits.

As seen in the VHDL code, after instantiating the three sub-units, their inputs are tied directly to the ALU's inputs (save for the `FUNC`s, which are tied to `FUNC_SEL`), and outputs are selectively tied to the ALU's outputs, dependent on `BLOCK_SEL`. As stated earlier, when an unsupported block is selected, the zero register (defined in the code as `ZeroVec`) is returned.

--- TESTBENCH ---

Initially, the testbench was written to permute all possible combinations of ***A***, ***B***, C_i and ***FUNC***. This implementation wasted a large amount of time cycling through unused values of ***FUNC***, and was hard to navigate the valid values due to the sheer quantity ***A*** and ***B*** values. The produced waveform was long and difficult to read, and thus was not that useful for verifying the functionality of the ALU design.

Therefore, instead of checking every single possible value, a testbench was created to cycle through all values of ***FUNC*** and C_i with random constant values of ***A*** and ***B***. This way it can easily be seen that each function of the ALU works as expected.

The VHDL code for the testbench can be found in **Appendix D**.

--- CONSTRAINTS ---

The constraints file can be found in **Appendix E**. Switches 0 to 3 were used for the four input bits of ***A*** and switches 4 to 7 for the input bits of ***B***. The four bits of ***F*** were mapped to switches 12 to 15 for readability. The final input, C_{in} was set to the center push button, or BTNC. For outputs, LEDs 0 to 3 were used to display the four bits of ***R***. Then ***N***, ***Z***, C_{out}, and ***V*** were set to LEDs 11, 10, 9, and 8 respectively.

SIMULATION RESULTS

Figure 1 Simulation Results for FUNC = “0000” to FUNC = “1000”

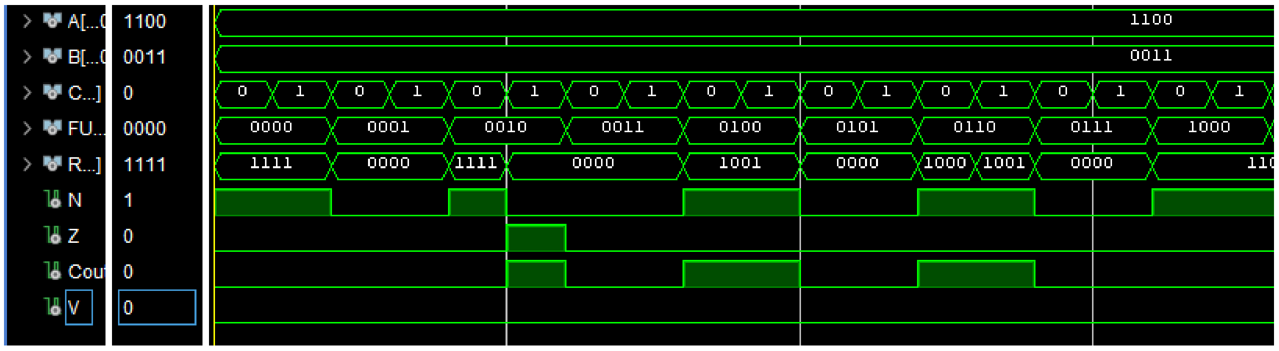
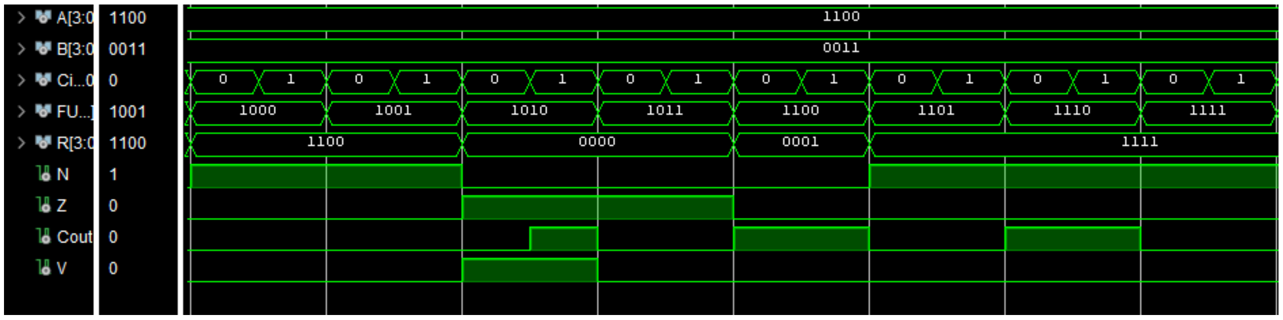


Figure 2 Simulation Results for FUNC = “1000” to FUNC = “1111”



SYNTHESIS RESULTS

Figure 3 Synthesis of Bitwise XOR

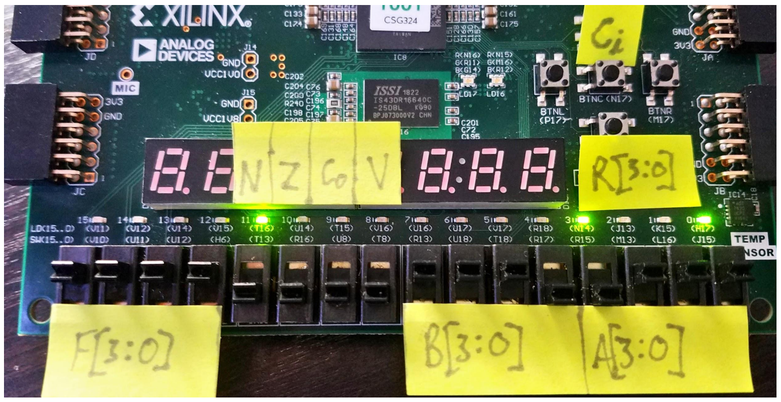


Figure 4 Synthesis of Add with Carry

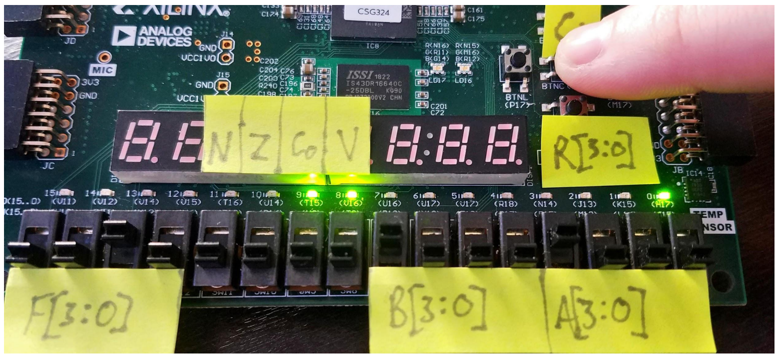
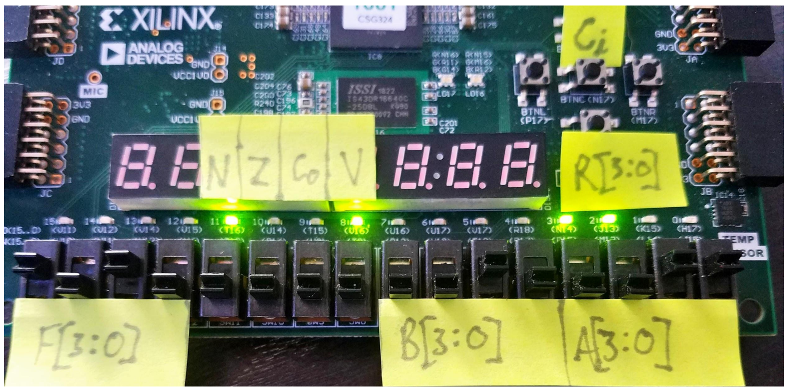


Figure 5 Synthesis of LSL



ISSUES ENCOUNTERED

The only issue of note that we encountered was that the barrel shifter from **Lab 4** took a single input `N_Layers`, which determined the shifter's `BUS_SIZE` internally. While this made sense previously, when using the shifter for the ALU, this design choice made it rather confusing what was happening internally. To fix this, the barrel shifter was slightly modified to take both `N_LAYERS` and `BUS_SIZE`. However, as `N_LAYERS` needed to conform to $log_2(BUS\_SIZE)$, we needed to use `IEEE.MATH._REAL` and `IEEE.NUMERIC_STD` libraries to generically compute the number of layers (held in the constant `SHIFTER_LAYERS` in the ALU VHDL code in **Appendix A**), as the log_2 operation was not supported by default.

We also encountered an issue where our ***R*** value for the ALU was incorrectly being set to **X** (undefined). The cause of this was quickly identified to be ***R*** being incorrectly tied directly to a sub-unit's output, rather than that unit's respective output signal occupying the output port.

CONCLUSIONS

Thanks to our previous testing and functionality verification of the adder/subtractor and barrel shifter, designing a functionally ALU was as simple as including a library. Had those units been dysfunctional or not been written in a generalizable and maintainable way, we would have spent many more hours than needed completing this lab.

While we did not test every possible input combination to the ALU, we believe our test data is random and representative enough to prove it's functionality.

APPENDIX A: ALU VHDL Code (alu.vhdl)

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

entity alu is
  generic(BUS_SIZE: natural :=1);
  Port (
  A, B: in std_logic_vector(BUS_SIZE-1 downto 0);
  F: in std_logic_vector(3 downto 0);
  Cin: in std_logic;
  R: out std_logic_vector(BUS_SIZE-1 downto 0);
  N,Z,Cout,V: out std_logic
  );
end alu;

architecture behavioral of alu is
    signal BLOCK_SEL: std_logic_vector(1 downto 0);
    signal FUNC_SEL: std_logic_vector(1 downto 0);
    signal N_mux,Z_mux,Cout_mux,V_mux : std_logic_vector(2
downto 0);
    signal SUM, SHIFTED, BITWISE : std_logic_vector(BUS_SIZE-1
downto 0);
    signal ZeroVec : STD_LOGIC_VECTOR(BUS_SIZE-1 downto 0);
    constant SHIFTER_LAYERS: natural :=
natural(ceil(log2(real(BUS_SIZE))));
    signal SHFT_AMT : STD_LOGIC_VECTOR(SHIFTER_LAYERS-1 downto
0);
begin
    ZeroVec <= (others => '0');
    FUNC_SEL <= F(2 downto 1);
    BLOCK_SEL <= F(3) & F(0);


    addsub: entity work.addsub (behavioral)
        generic map(BUS_SIZE=>BUS_SIZE)
        port
map(A=>A,B=>B,Cin=>Cin,S=>SUM,N=>N_mux(0),Z=>Z_mux(0),V=>V_mux(
0),Cout=>Cout_mux(0),FUNC=>FUNC_SEL);
```

```
    logic_unit: entity work.logic_unit (behavioral)
        generic map(BUS_SIZE=>BUS_SIZE)
        port map(A=>A,B=>B,Y=>BITWISE, N=>N_mux(1),
Z=>Z_mux(1), FUNC=>FUNC_SEL);
        V_mux(1) <= '0';
        Cout_mux(1) <= '0';
    shifter: entity work.shifter (behavioral)
        generic map(BUS_SIZE=>BUS_SIZE,
N_LAYERS=>SHIFTER_LAYERS)
        port map(rIN=>A, rOUT=>SHIFTED,
shftAmnt=>B(SHIFTER_LAYERS-1 downto
0),N=>N_mux(2),Z=>Z_mux(2),V=>V_mux(2),Cout=>Cout_mux(2),FUNC=>
FUNC_SEL);
        SHFT_AMT <= B(SHIFTER_LAYERS-1 downto 0);



   with BLOCK_SEL select
    R <= SUM     when "00", --ADD/SUB
         BITWISE when "11", --Logic
         SHIFTED when "10", --Shifter
         ZeroVec when others; --Reserved (Return 0)

    with BLOCK_SEL select
    N <= N_mux(0)   when "00", --ADD/SUB
         N_mux(1)   when "11", --Logic
         N_mux(2)   when "10", --Shifter
         '0'        when others; --Reserved (Return 0)

    with BLOCK_SEL select
    Z <= Z_mux(0)   when "00", --ADD/SUB
         Z_mux(1)   when "11", --Logic
         Z_mux(2)   when "10", --Shifter
         '0'        when others; --Reserved (Return 0)

    with BLOCK_SEL select
    V <= V_mux(0)     when "00", --ADD/SUB
         V_mux(1)  when "11", --Logic
         V_mux(2)   when "10", --Shifter
         '0'  when others; --Reserved (Return 0)

    with BLOCK_SEL select
    Cout <= Cout_mux(0) when "00", --ADD/SUB
```

```
         '0'              when "11", --Logic
         Cout_mux(0)      when "10", --Shifter
         '0'              when others; --Reserved (Return 0)

end behavioral;
```

APPENDIX B: Logic Unit VHDL Code (logic_unit.vhdl)

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

entity logic_unit is
  generic(BUS_SIZE: natural :=1);
  Port (
  A, B: in std_logic_vector(BUS_SIZE-1 downto 0);
  Y: out std_logic_vector(BUS_SIZE-1 downto 0);
  N,Z: out std_logic;
  FUNC: in std_logic_vector(1 downto 0)
  );
end logic_unit;

architecture behavioral of logic_unit is
  signal R: std_logic_vector(BUS_SIZE-1 downto 0);
begin

    with FUNC select
        R <= not(B)     when "00",
             (A and B)  when "01",
             (A or B)   when "10",
             (A xor B)  when others;

    Y <= R;
    N <= R(BUS_SIZE-1);
    Z <= '1' when (R = 0) else '0';
end behavioral;
```

APPENDIX C: Modified Barrel Shifter VHDL Code (shifter.vhdl)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.all;
use IEEE.NUMERIC_STD.ALL;

entity shifter is
    generic(BUS_SIZE: natural := 8;
            N_LAYERS: natural := 3); --N-LAYERS =
log2(BUS_SIZE)
    port(rIN : in STD_LOGIC_VECTOR(BUS_SIZE-1 downto 0);
         rOUT : out STD_LOGIC_VECTOR(BUS_SIZE-1 downto 0);
         shftAmnt : in STD_LOGIC_VECTOR(N_LAYERS-1 downto 0);
         N,Z,V,Cout : out STD_LOGIC;
         FUNC : in STD_LOGIC_VECTOR (1 downto 0)
         );
end shifter;

architecture behavioral of shifter is
    type STD_LOGIC_VECTOR_2 is array (integer range<>) of
STD_LOGIC_VECTOR(BUS_SIZE-1 downto 0);
    signal A : STD_LOGIC_VECTOR_2(N_LAYERS-1 downto 0);
    signal B : STD_LOGIC_VECTOR_2(N_LAYERS-1 downto 0);
    signal Y : STD_LOGIC_VECTOR_2(N_LAYERS-1 downto -1);
    signal ZeroVec : STD_LOGIC_VECTOR(BUS_SIZE-1 downto 0);
    signal OneVec : STD_LOGIC_VECTOR(BUS_SIZE-1 downto 0);
    constant NOP: STD_LOGIC_VECTOR(1 downto 0) := "00";
    constant LSL: STD_LOGIC_VECTOR(1 downto 0) := "01";
    constant LSR: STD_LOGIC_VECTOR(1 downto 0) := "10";
    constant ASR: STD_LOGIC_VECTOR(1 downto 0) := "11";
begin
    ZeroVec <= (others => '0');
    OneVec <= (others => '1');
    Y(-1) <= rIN;

    muxes: for layer in N_LAYERS-1 downto 0 generate
        muxN: entity work.mux (behavioral)
            generic map(BUS_SIZE=>BUS_SIZE)
            port map(A=>A(layer),
                     B=> B(layer),
                     SEL=>shftAmnt(layer),
                     Y=>Y(layer));
        A(layer) <= Y(layer-1);
        B(layer) <= Y(layer-1)(BUS_SIZE-1-2**layer downto 0) &
ZeroVec(2**layer-1 downto 0) when FUNC=LSL else
                ZeroVec(2**layer-1 downto 0) &
Y(layer-1)(BUS_SIZE-1 downto 2**layer) when FUNC=LSR else
                ZeroVec(2**layer-1 downto 0) &
Y(layer-1)(BUS_SIZE-1 downto 2**layer) when (FUNC=ASR and
rIN(BUS_SIZE-1)='0') else
                OneVec(2**layer-1 downto 0) &
Y(layer-1)(BUS_SIZE-1 downto 2**layer) when (FUNC=ASR and
rIN(BUS_SIZE-1)='1') else
                A(layer); --No shift
    end generate muxes;


    rOUT <= Y(N_LAYERS-1);
    N <= Y(N_LAYERS-1)(BUS_SIZE-1);
    Z <= '1' when Y(N_LAYERS-1) = 0 else '0';
    V <= '1' when Y(N_LAYERS-1)(BUS_SIZE-1) /= rIN(BUS_SIZE-1)
AND FUNC=LSL else '0';
    Cout <= rIN(BUS_SIZE - 1 - to_integer(unsigned(shftAmnt)))
when shftAmnt /= 0 AND FUNC /= NOP else '0';
end behavioral;
```

APPENDIX D: ALU Testbench VHDL Code (alu_testbench.vhdl)

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

entity alu_testbench is
end alu_testbench;

architecture Behavioral of alu_testbench is
constant BUS_SIZE: natural := 4;
signal A, B, R: std_logic_vector(BUS_SIZE-1 downto 0);
signal FUNC : std_logic_vector(3 downto 0);
signal Cin : std_logic_vector(0 to 0);
signal N,Z,Cout,V: std_logic;
begin

ALU: entity work.alu (behavioral)
    generic map(BUS_SIZE=>BUS_SIZE)
    port map(A=>A,B=>B,F=>FUNC,Cin=>Cin(0),
             R=>R,N=>N,Z=>Z,V=>V,Cout=>Cout);

test: process
variable i, j, k: integer;
begin

A <= "1100";
B <= "0011";
FUNC <= (others => '0');
Cin <= (others => '0');

for i in 0 to 15 loop
    Cin <= "0";
    FUNC <= std_logic_vector(TO_UNSIGNED(i,4));
    wait for 1ns;
    Cin <= "1";
    wait for 1ns;
end loop;
end process;
end Behavioral;
```

APPENDIX E: Contains File (alu_constraints.xdc)

```
##Switches
set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 }
[get_ports { A[0] }];
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 }
[get_ports { A[1] }];
set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 }
[get_ports { A[2] }];
set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 }
[get_ports { A[3] }];
set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 }
[get_ports { B[0] }];
set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 }
[get_ports { B[1] }];
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 }
[get_ports { B[2] }];
set_property -dict { PACKAGE_PIN R13   IOSTANDARD LVCMOS33 }
[get_ports { B[3] }];
set_property -dict { PACKAGE_PIN H6    IOSTANDARD LVCMOS33 }
[get_ports { F[0] }];
set_property -dict { PACKAGE_PIN U12   IOSTANDARD LVCMOS33 }
[get_ports { F[1] }];
set_property -dict { PACKAGE_PIN U11   IOSTANDARD LVCMOS33 }
[get_ports { F[2] }];
set_property -dict { PACKAGE_PIN V10   IOSTANDARD LVCMOS33 }
[get_ports { F[3] }];

## LEDs
set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 }
[get_ports { R[0] }];
set_property -dict { PACKAGE_PIN K15   IOSTANDARD LVCMOS33 }
[get_ports { R[1] }];
set_property -dict { PACKAGE_PIN J13   IOSTANDARD LVCMOS33 }
[get_ports { R[2] }];
set_property -dict { PACKAGE_PIN N14   IOSTANDARD LVCMOS33 }
[get_ports { R[3] }];
set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 }
[get_ports { V }];
```

```
set_property -dict { PACKAGE_PIN T15   IOSTANDARD LVCMOS33 }
[get_ports { Cout }];
set_property -dict { PACKAGE_PIN U14   IOSTANDARD LVCMOS33 }
[get_ports { Z }];
set_property -dict { PACKAGE_PIN T16   IOSTANDARD LVCMOS33 }
[get_ports { N }];

## Buttons
set_property -dict { PACKAGE_PIN N17   IOSTANDARD LVCMOS33 }
[get_ports { Cin }];
```