2020-2021 Fall Semester

Bilkent University

EEE-102 Term Project Final Report

Student Name: Muhammed Şamil Arınç

Student ID: 21803683

Project Name: Chameleon Robot

Instructor: Cem Tekin

Section: 2

YouTube Link: youtu.be/dD3BZdNYPnQ

Abstract

The main purposes of this term project are designing a robot that looks and behaves like a chameleon in some aspects, writing a VHDL code that gives this robot some abilities and implementing this code to FPGA board called BASYS-3. This robot can be the basic step of making a robot that can hide from radars and people's eyes by using more advanced reflectors.

Project Design

The robot will behave like a chameleon. It will scan the surface color periodically and return the color value that it obtained from surface with the color sensor. It also supplemented by a led placed on the robot's bottom part and change the color of the displays that I will put on the chameleon robot's back. It will also be able to move by its legs according to the Bluetooth signal that will be generated by a mobile phone app.

Design Methodology

Tools that will be used in this project are:

- BASYS-3 Board
- Mobile Phone
- #4 - Common Anode RGB Leds
- #2 - TCS-230 Color Sensors
- #1 - HC-05 Bluetooth Module
- #4 - Servo Motors
- Arduino Mega
- Many Jumpers
- Breadboard
- #2 – 220Ω Resistors
- Android Bluetooth App

The main working mechanisms of the robot's color changing algorithm is as follows: first it reads the color from color sensors, then it changes the color of the RGB leds placed on it by giving them 3 outputs (Red, Green and Blue).

Top Module (1A): I designed this module to connect my Bluetooth module and servo module. I designed my robot such that it moves according to the data came from Bluetooth.

Bluetooth Top Module (1B): In this module, I connected RX and TX modules of my Bluetooth module and transfer Bluetooth data to top module in order to use it with servo modules.

Bluetooth RX Module (1C): In this module, I made a Finite State Machine to read 1-byte data with 8 bits. When it takes the start bit, the FSM initiates and it works until reaching the stop bit as: The FSM's transfer state initiates with starter signal which is '0' and lasts 8 clock periods to take 1-byte (8 bits). While signal is '1', the process does not start. After taking 8 bits, it waits for '1' which is the stop bit. If it takes '0' as stop bit, it does not use the data transferred and return to 1st (Idle) state. When stop signal is true, it validates the data and returns the 1-byte data to top module in order to use it in servo module.

Bluetooth TX Module (1F): In the first hand the TX module is necessary to read different feedbacks of Bluetooth module with different data inputs; however, after learning the algorithm HC-05 uses, I disabled TX module and comment the codes which are about this module.

Left Servo Module (1D): I implemented this module to move my robot's left leg. The movement of my chameleon robot's legs are as follows: First it moves the part of its leg under the knee to step forward, second it moves the legs upper part to up so that its leg is no longer touches the ground. Third, it moves the lower part of the leg to backward. At last, it moves the legs upper part to down, so that it reaches its initial position again. I used Finite State Machine to implement this movement to VHDL code. The servo motors work with a special digital signal format "Pulse with Modulation (PWM)". I implemented this pulse with a process in my code. For SG90 servos the pwm period lasts 20ms. The pulse's duration is called "duty cycle" and it lasts 1 to 2 milliseconds. For 1ms duty cycle, servo motor moves to 0^{o} and for 2ms duty cycle it moves 180^{o}. For the values between 1-2ms, the servo takes different angle values. For my design, special data came from Bluetooth module initiates the movement of the left leg.

Right Servo Module (1E): It is very similar to the left servo design with some differences. I changed the direction of the rotation and changed the Bluetooth command which initiates servo's movement.

Arduino Color Sensor and RGB Controller (1G): In this code, I took inputs from color sensors and use them with my RGB leds.

Color sensors sets its outputs S2 and S3 outputs as:

LOW-LOW to read red

LOW-HIGH to read blue

HIGH-HIGH to read green

After setting output, it takes PWM signals and read them as 8-bit color codes. These codes are between 0 and 255 for each of 3 colors. After reading colors, the program works with RGB leds.

The common anode RGB leds has 4 legs: Red, Green, Blue and V_{cc}. It also works with PWM signals. I sent the data read from color sensors to leds and display the color of the ground on top of my robot's design.

Constraints File (1H): In this file, I connected the top module's ports to Basys3 board's Pmod Headers, buttons and clock. I used Pmod Headers to Rx input, Tx output and pwm outputs of each 4 servo motors.

Bluetooth App for Android Devices: I designed a Bluetooth App to control my chameleon robot using MIT App Inventor designing tool. In my design I used 3 buttons: move left leg, move right leg and stop. I also uploaded my APK file to Google Drive: https://drive.google.com/file/d/18wNobDDO37FYa4zFW3IFOKQ-AWmB0mS-/view?usp=sharing. Figure 1 shows the blocks view of my app inventor design.

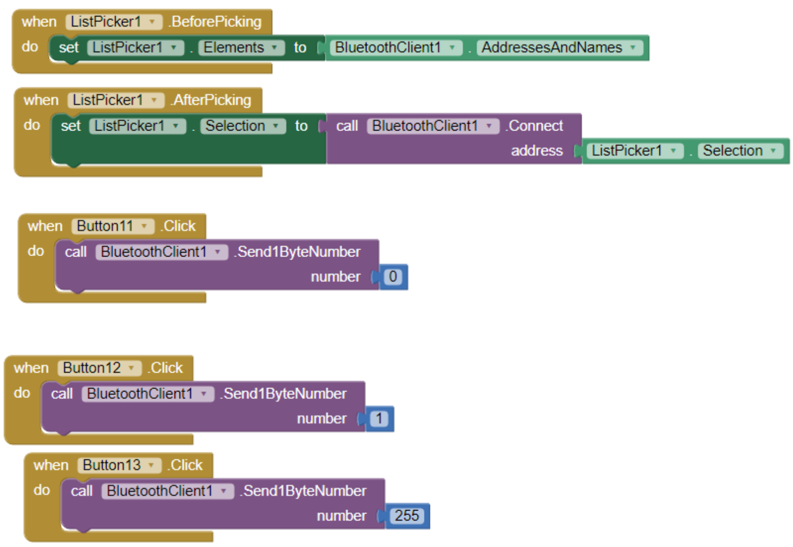

Figure 1

Results

It can be seen on my RTL design (Figure 2) that I implement 3 inputs: clk, reset, rx and 5 outputs: pwmSignal1, pwmSignal2, pwmSignal3, pwmSignal4, tx. For the special cases of Bluetooth data, it gives rightCommand as '1' to right sided servo motors, or leftCommand as '1' to left sided servo motors. Also, Figure 3 shows the RTL schematic of my Servo module and Figure 4 shows my Bluetooth module. I also add a screenshot of the Bluetooth application that I designed to control my Chameleon Robot.

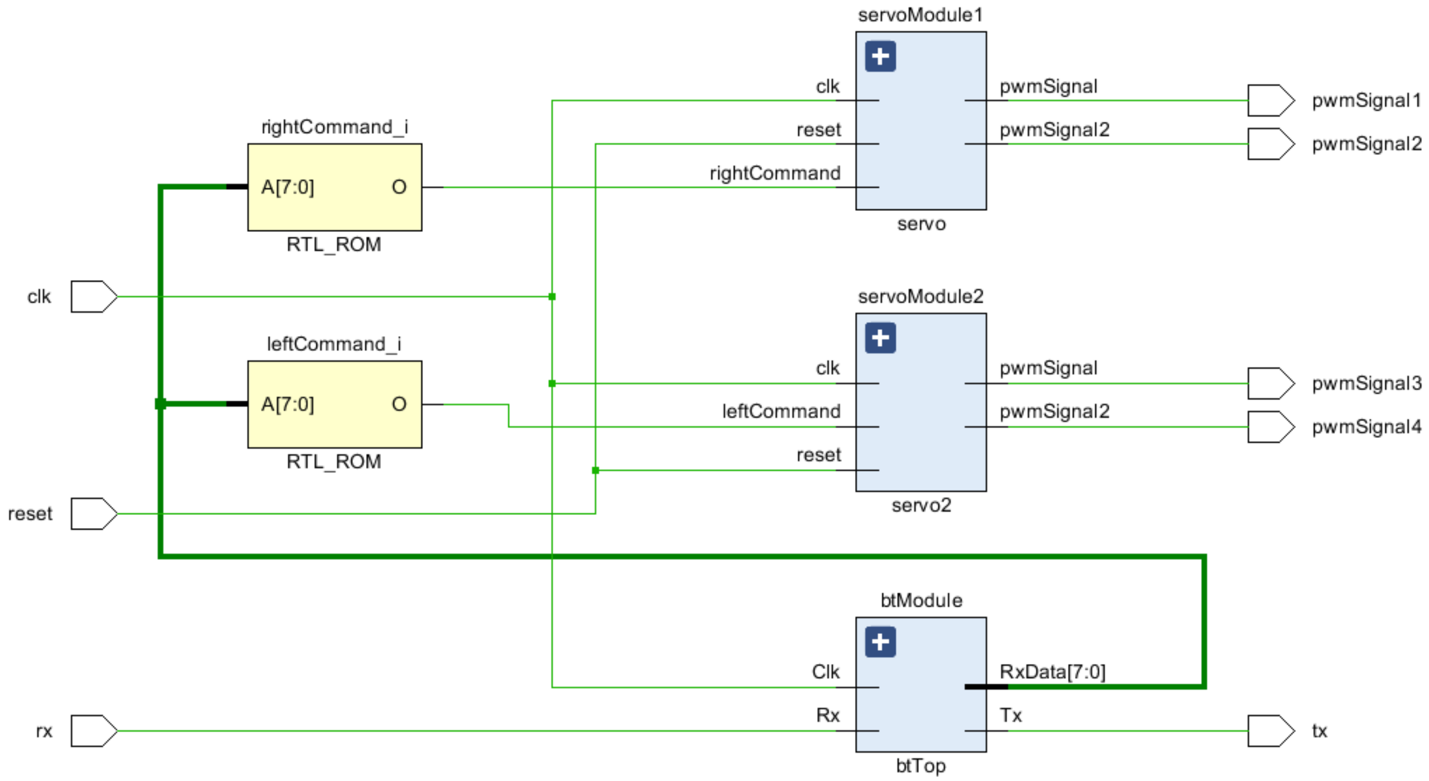

Figure 2

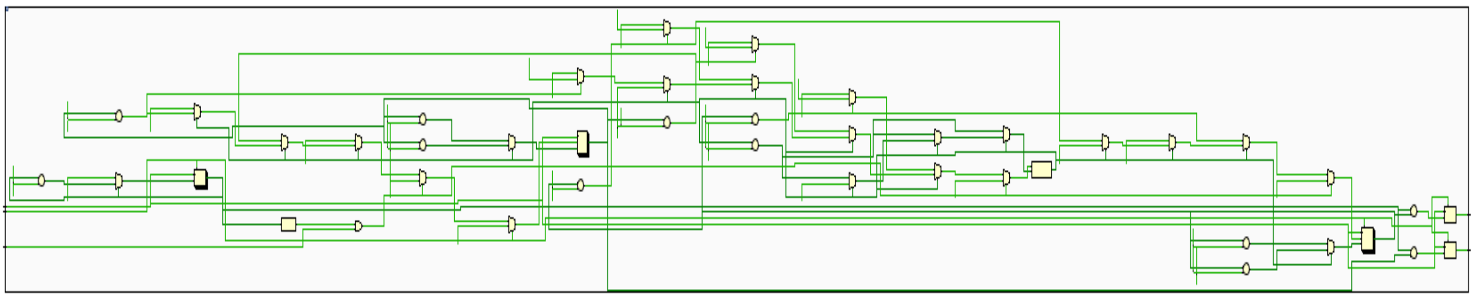

Figure 3

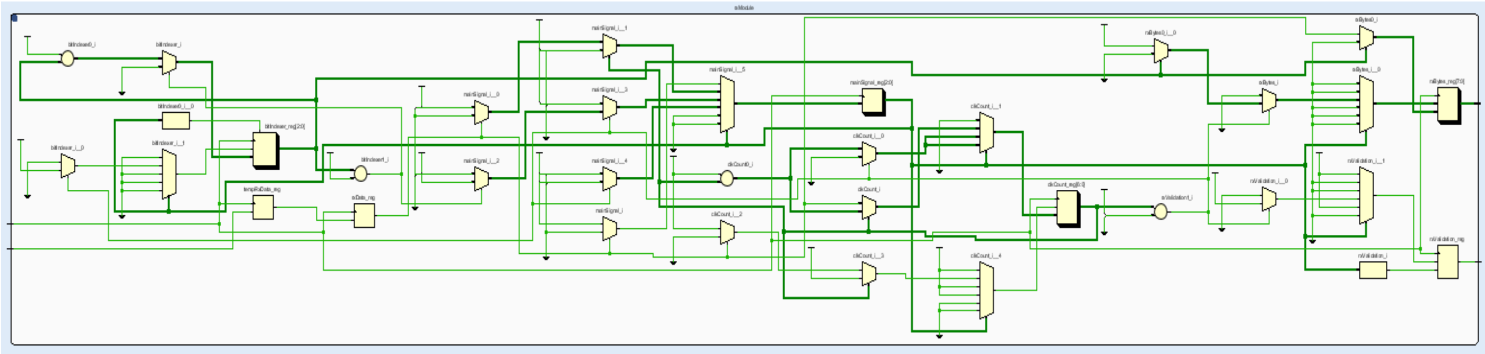

Figure 4

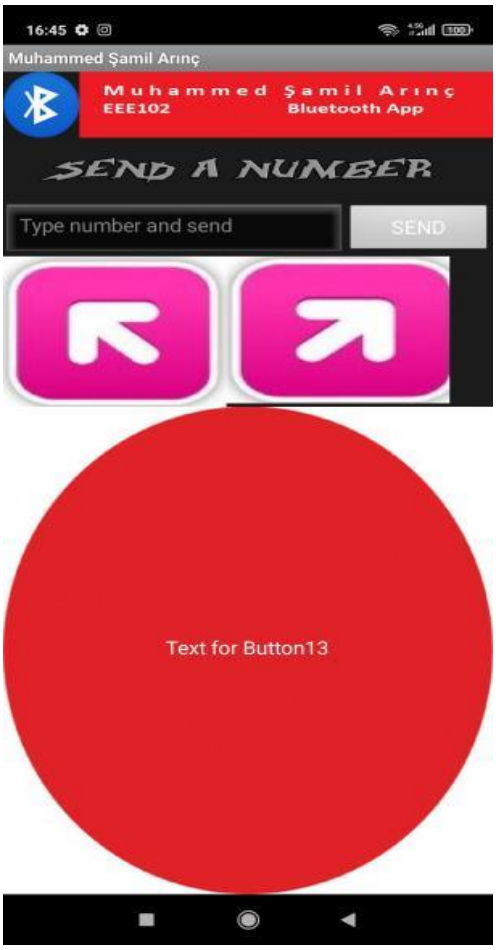

Figure 5

Hardware Demo

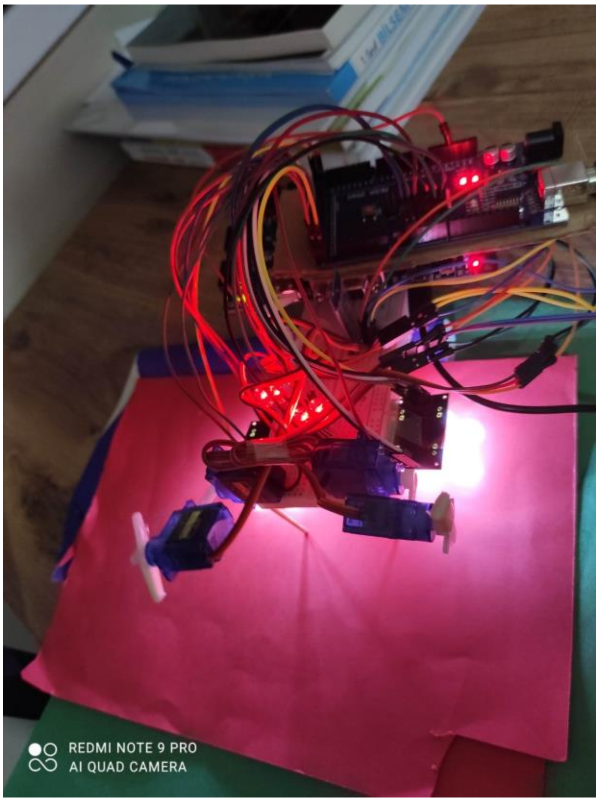

Figure 6

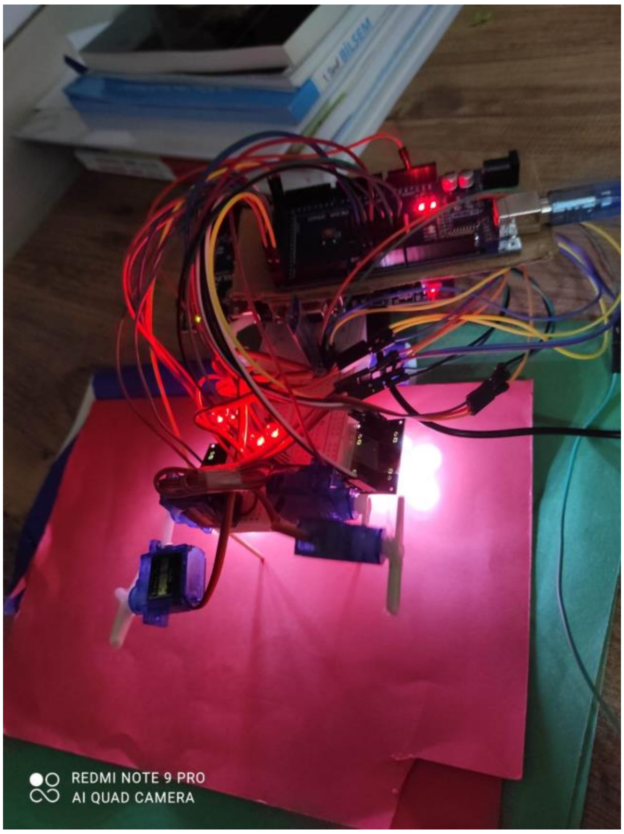

Figure 7

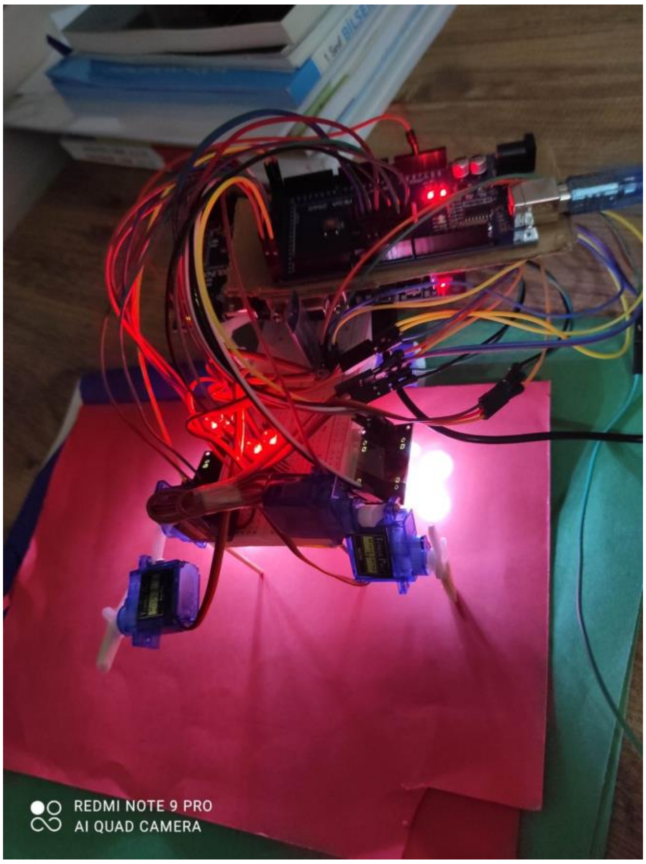

Figure 8

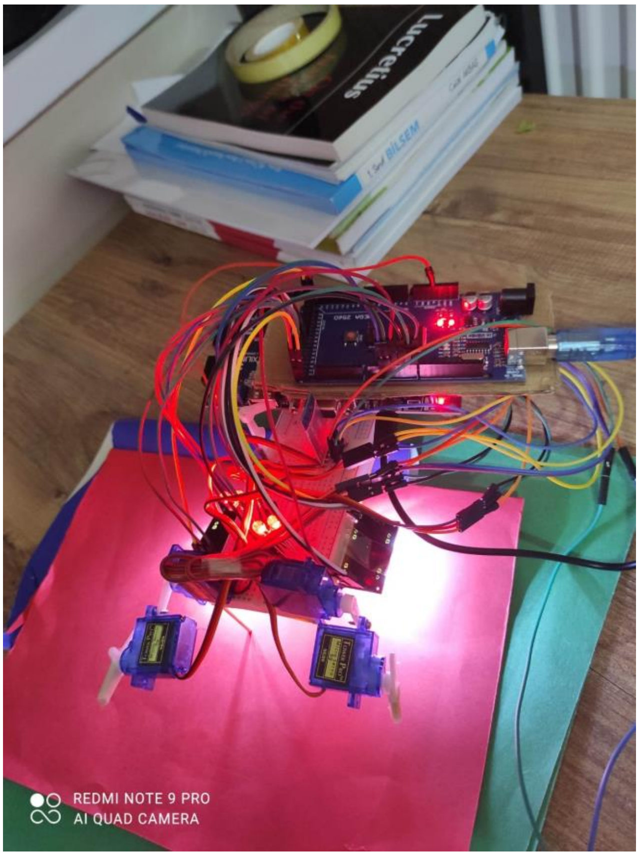

Figure 9

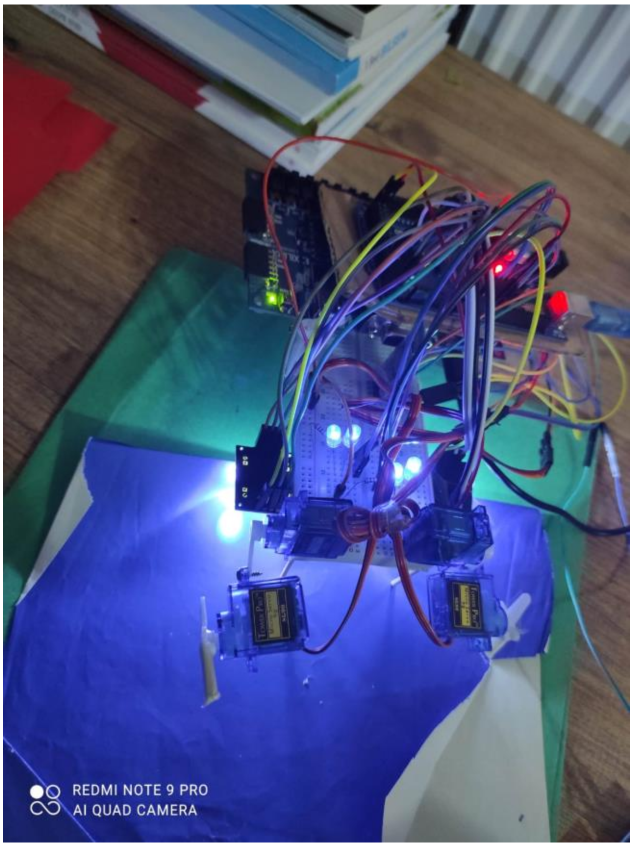

Figure 10

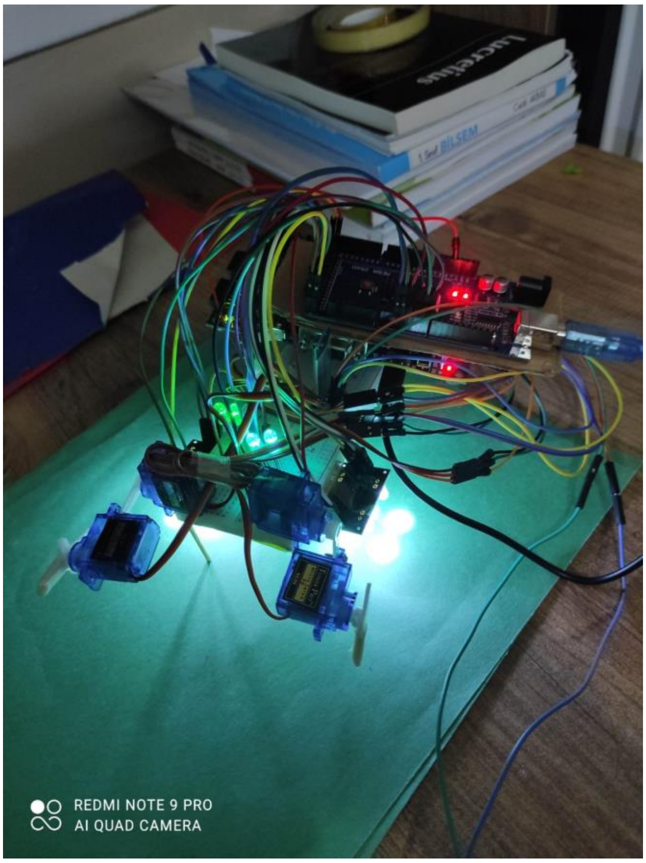

Figure 11

References

Basys 3™ FPGA Board Reference Manual. (2016, April 8). Retrieved October 16, 2020, from https://reference.digilentinc.com/_media/basys3:basys3_rm.pdf.

FPGA ile RC Servo Motor Kontrolü. Retrieved December 25, 2020, from https://roboturka.com.

UART, Serial Port, RS-232 Interface. Retrieved December 25, 2020, from https://www.nandland.com.

Appendix

1A: Top Module

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

entity topModule is
     Port (    clk, rx, reset : in std_logic;
               tx, pwmSignal1, pwmSignal2, pwmSignal3, pwmSignal4 :
out std_logic
            );
  end topModule;

architecture Behavioral of topModule is
component btTop
     Port(   Clk, Rx : in std_logic;
             RxData:  out std_logic_vector(7 downto 0);
             Tx: out std_logic
     );
  end component;

component servo is
     Port (   clk            : in   STD_LOGIC;
              reset          : in   STD_LOGIC;
              rightCommand   : in   STD_LOGIC;
              pwmSignal      : out  STD_LOGIC;
              pwmSignal2     : out  STD_LOGIC
              );
  end component;

component servo2 is
```

```
      Port (   clk            : in   STD_LOGIC;
               reset          : in   STD_LOGIC;
               leftCommand    : in   STD_LOGIC;
               pwmSignal      : out  STD_LOGIC;
               pwmSignal2     : out  STD_LOGIC
               );
 end component;

signal rightCommand : std_logic;
signal leftCommand : std_logic;
signal data : std_logic_vector(7 downto 0) := "10000000";

begin
rightCommand <= '1' when data = "11111111" else '0';
leftCommand <= '1' when data = "00000000" else '0';
btModule : btTop port map(
    clk => clk,
    rx => rx,
    tx => tx,
    RxData => data
);

servoModule1 : servo port map(
    clk => clk,
    reset => reset,
    rightCommand => rightCommand,
    pwmSignal => pwmSignal1,
    pwmSignal2 => pwmSignal2
);

servoModule2 : servo2 port map(
    clk => clk,
    reset => reset,
```

```
    leftCommand => leftCommand,
    pwmSignal => pwmSignal3,
    pwmSignal2 => pwmSignal4
);
end Behavioral;
```

1B: Bluetooth Top Module

```
library ieee;
use ieee.std_logic_1164.ALL;
use ieee.numeric_std.all;

entity btTop is
port(   Clk, Rx : in std_logic;
        RxData:  out std_logic_vector(7 downto 0);
        Tx: out std_logic
);
end btTop;

architecture Behavioral of btTop is

--component btTx is
--  generic (
--    clocksPerBits : integer := 115
--    );
--  port (
--    clk       : in  std_logic;
--    txDivisor     : in  std_logic;
--    txBytesTransferred   : in  std_logic_vector(7 downto 0);
--    txActivated : out std_logic;
```

```
--    txSerialConnection : out std_logic;
--    txFinished   : out std_logic
--    );
--end component;

component btRx is
  generic (
    clocksPerBits : integer := 115     -- Needs to be set correctly
    );
  port (
    clk       : in  std_logic;
    rxSerialConnection : in  std_logic;
    rxDivisor     : out std_logic;
    rxBytesTransferred   : out std_logic_vector(7 downto 0)
    );
end component;

-- Basys3 clock hızı = 100 MHz
-- BT modülü 115200 baud rate
-- 10000000 / 115200 = 87
constant clocksPerBits : integer := 87;

--  constant bitPeriod : time := 8680 ns;

--  signal txDivisor     : std_logic                    := '0';
--  signal txBytesTransferred   : std_logic_vector(7 downto 0) :=
(others => '0');
--  signal txSerialConnection : std_logic;
--  signal txFinished   : std_logic;



begin
```

```
--testBench sonuçlarını görmek için
--  txModule : btTx
--    generic map (
--      clocksPerBits => clocksPerBits
--      )
--    port map (
--      clk        => clk,
--      txDivisor    => txDivisor,
--      txBytesTransferred   => txBytesTransferred,
--      txActivated => open,
--      txSerialConnection => txSerialConnection,
--      txFinished   => txFinished
--      );


rxModule : btRx
    generic map (
      clocksPerBits => clocksPerBits
      )
    port map (
      clk       => clk,
      rxSerialConnection => rx,
      rxDivisor => tx,
      rxBytesTransferred   => RxData
      );


end Behavioral;
```

1C: Bluetooth RX Module

```
library ieee;
use ieee.std_logic_1164.ALL;
use ieee.numeric_std.all;

entity btRx is
  generic (
    clocksPerBits : integer := 115
    );
  port (
    clk       : in  std_logic;
    rxSerialConnection : in  std_logic;
    rxDivisor     : out std_logic;
    rxBytesTransferred   : out std_logic_vector(7 downto 0)
    );
end btRx;


architecture Behavioral of btRx is

  type rxType is (idleSignal, startingBit, dataTransferred,
                  stopperBit, cleanAll);
  signal mainSignal : rxType := idleSignal;

  signal tempRxData : std_logic := '0';
  signal rxData   : std_logic := '0';

  signal clkCount : integer range 0 to clocksPerBits-1 := 0;
  signal bitIndexer : integer range 0 to 7 := 0;  -- 8 Bits Total
  signal rxBytes   : std_logic_vector(7 downto 0) := (others =>
'0');
  signal rxValidation     : std_logic := '0';

begin
```

```
doubleVeri : process (clk)
begin
  if rising_edge(clk) then
    tempRxData <= rxSerialConnection;
    rxData   <= tempRxData;
  end if;
end process;

btFSM : process (clk)
begin
  if rising_edge(clk) then

    case mainSignal is

      when idleSignal =>
        rxValidation     <= '0';
        clkCount <= 0;
        bitIndexer <= 0;

        if rxData = '0' then        -- Start bit
          mainSignal <= startingBit;
        else
          mainSignal <= idleSignal;
        end if;

      when startingBit =>
        if clkCount = (clocksPerBits-1)/2 then
          if rxData = '0' then
            clkCount <= 0;
            mainSignal   <= dataTransferred;
          else
            mainSignal   <= idleSignal;
```

```
    end if;
  else
    clkCount <= clkCount + 1;
    mainSignal   <= startingBit;
  end if;

when dataTransferred =>
  if clkCount < clocksPerBits-1 then
    clkCount <= clkCount + 1;
    mainSignal   <= dataTransferred;
  else
    clkCount           <= 0;
    rxBytes(bitIndexer) <= rxData;

    if bitIndexer < 7 then
      bitIndexer <= bitIndexer + 1;
      mainSignal   <= dataTransferred;
    else
      bitIndexer <= 0;
      mainSignal   <= stopperBit;
    end if;
  end if;



-- Stop bit = 1
when stopperBit =>
  if clkCount < clocksPerBits-1 then
    clkCount <= clkCount + 1;
    mainSignal   <= stopperBit;
  else
    rxValidation     <= '1';
    clkCount <= 0;
    mainSignal   <= cleanAll;
```

```
          end if;

        when cleanAll =>
          mainSignal <= idleSignal;
          rxValidation   <= '0';


        when others =>
          mainSignal <= idleSignal;

      end case;
    end if;
  end process;

  rxDivisor   <= rxValidation;
  rxBytesTransferred <= rxBytes;

end Behavioral;
```

1D: Servo Module for Motors on Left

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

entity servo is
  Port ( clk           : in   STD_LOGIC;
         reset         : in   STD_LOGIC;
         rightCommand  : in   STD_LOGIC;
         pwmSignal     : out  STD_LOGIC;
         pwmSignal2    : out  STD_LOGIC
         );
```

```
end servo;

architecture Behavioral of servo is

constant speed              :   integer :=       500;
constant period             :   integer :=   1000000;
constant max_duty           :   integer :=    250000;
constant min_duty           :   integer :=     50000;

signal pwm                  :   std_logic;
signal pwmTemp              :   std_logic;
signal pwm2                 :   std_logic;
signal pwmTemp2             :   std_logic;
signal flag                 :   std_logic;
signal lenSignal            :   integer :=            150000;
signal lenSignalTemp        :   integer :=            150000;
signal lenSignal2           :   integer :=            150000;
signal lenSignalTemp2       :   integer :=            150000;
signal counter              :   integer :=            0;
signal counterTemp          :   integer :=            0;
signal state                :   integer :=            0;
signal resetter             :   integer :=            0;

constant legBack            :   integer :=       50000;
constant legFront           :   integer :=      200000;
constant kneeUp             :   integer :=       72000;
constant kneeDown           :   integer :=      175000;

begin
process (clk, reset)
  begin
    if reset = '1' then
        pwm <= '0';
```

```
        pwm2 <= '0';
        counter <= 0;
        lenSignal <= 0;

    elsif clk='1' and clk'event then
        pwm <= pwmTemp;
        pwm2 <= pwmTemp2;
        counter <= counterTemp;
        lenSignal <= lenSignalTemp;
        lenSignal2 <= lenSignalTemp2;
      end if;

  end process;

counterTemp <= 0 when counter = period else counter + 1;
flag <= '1' when counter = 0 else '0';

process(rightCommand, flag, lenSignal)
  begin
    lenSignalTemp <= lenSignal;
    lenSignalTemp2 <= lenSignal2;
    if flag='1' and rightCommand = '1' then
        if state = 0 then
                if lenSignal > legBack then
                    lenSignalTemp <= lenSignal - speed;
                else
                    resetter <= resetter + 1;
                  end if;
       elsif state = 1 then
                if lenSignal2 > kneeUp then
                    lenSignalTemp2 <= lenSignal2 - speed;
                else
                    resetter <= resetter + 1;
```

```
                  end if;
      elsif state = 2 then
                 if lenSignal < legFront then
                     lenSignalTemp <= lenSignal + speed;
                 else
                     resetter <= resetter + 1;
                  end if;
      elsif state = 3 then
                if lenSignal2 < kneeDown then
                     lenSignalTemp2 <= lenSignal2 + speed;
                 else
                     resetter <= 0;
                  end if;
      else
                 if lenSignal2 < kneeDown then
                     lenSignalTemp2 <= lenSignal2 + speed;
                 else
--                     state <= 0;
                     resetter <= resetter;
                  end if;
         end if;
      end if;
end process;

--status <= state;
pwmSignal <= pwm;
pwmSignal2 <= pwm2;
pwmTemp <= '1' when counter < lenSignal else '0';
pwmTemp2 <= '1' when counter < lenSignal2 else '0';
state <= resetter;

end Behavioral;
```

1E: Servo Module for Motors on Right

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

entity  servo2 is
  Port ( clk           : in   STD_LOGIC;
         reset         : in   STD_LOGIC;
         leftCommand   : in   STD_LOGIC;
         pwmSignal     : out  STD_LOGIC;
         pwmSignal2    : out  STD_LOGIC
         );
end  servo2;

architecture Behavioral of servo2 is

constant speed              :   integer :=       500;
constant period             :   integer :=   1000000;
constant max_duty           :   integer :=    250000;
constant min_duty           :   integer :=     50000;

signal pwm                  :   std_logic;
signal pwmTemp              :   std_logic;
signal pwm2                 :   std_logic;
signal pwmTemp2             :   std_logic;
signal flag                 :   std_logic;
signal lenSignal            :   integer :=            150000;
signal lenSignalTemp        :   integer :=            150000;
signal lenSignal2           :   integer :=            150000;
signal lenSignalTemp2       :   integer :=            150000;
```

```
signal counter            :   integer :=           0;
signal counterTemp        :   integer :=           0;
signal state              :   integer :=           0;
signal resetter           :   integer :=           0;

constant legBack          :   integer :=       50000;
constant legFront         :   integer :=      200000;
constant kneeUp           :   integer :=       72000;
constant kneeDown         :   integer :=      175000;

begin
process(clk, reset)
  begin
    if reset = '1' then
        pwm <= '0';
        pwm2 <= '0';
        counter <= 0;
        lenSignal <= 0;

    elsif clk='1' and clk'event then
        pwm <= pwmTemp;
        pwm2 <= pwmTemp2;
        counter <= counterTemp;
        lenSignal <= lenSignalTemp;
        lenSignal2 <= lenSignalTemp2;
      end if;

  end process;

counterTemp <= 0 when counter = period else counter + 1;
flag <= '1' when counter = 0 else '0';

process(leftCommand, flag, lenSignal)
```

```
  begin
    lenSignalTemp <= lenSignal;
    lenSignalTemp2 <= lenSignal2;
    if flag='1' and leftCommand = '1' then
        if state = 0 then
                if lenSignal > legBack then
                    lenSignalTemp <= lenSignal - speed;
                else
                    resetter <= resetter + 1;
                  end if;
       elsif state = 1 then
                if lenSignal2 > kneeUp then
                    lenSignalTemp2 <= lenSignal2 - speed;
                else
                    resetter <= resetter + 1;
                  end if;
      elsif state = 2 then
                if lenSignal < legFront then
                    lenSignalTemp <= lenSignal + speed;
                else
                    resetter <= resetter + 1;
                  end if;
      elsif state = 3 then
               if lenSignal2 < kneeDown then
                    lenSignalTemp2 <= lenSignal2 + speed;
                else
                    resetter <= 0;
                  end if;
      else
                if lenSignal2 < kneeDown then
                    lenSignalTemp2 <= lenSignal2 + speed;
                else
--                      state <= 0;
```

```
                    resetter <= resetter;
                  end if;
         end if;
      end if;
end process;

--status <= state;
pwmSignal <= pwm;
pwmSignal2 <= pwm2;
pwmTemp <= '1' when counter < lenSignal else '0';
pwmTemp2 <= '1' when counter < lenSignal2 else '0';
state <= resetter;

end Behavioral;
```

1F: Bluetooth TX Module

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity btTx is
  generic (
    clocksPerBits : integer := 115     -- Needs to be set correctly
    );
  port (
    clk       : in  std_logic;
    txDivisor     : in  std_logic;
    txBytesTransferred   : in  std_logic_vector(7 downto 0);
    txActivated : out std_logic;
    txSerialConnection : out std_logic;
```

```
    txFinished   : out std_logic
    );
end btTx;


architecture Behavioral of btTx is

  type txType is (idleSignal, startingBit, dataTransferred,
                    stopperBit, cleanAll);
  signal mainSignal : txType := idleSignal;

  signal newClock : integer range 0 to clocksPerBits-1 := 0;
  signal bitIndexer : integer range 0 to 7 := 0;  -- 8 Bits Total
  signal txData   : std_logic_vector(7 downto 0) := (others => '0');
  signal txDone   : std_logic := '0';

begin


txProcess : process (clk)
  begin
    if rising_edge(clk) then

      case mainSignal is

        when idleSignal =>
          txActivated <= '0';
          txSerialConnection <= '1';
          txDone   <= '0';
          newClock <= 0;
          bitIndexer <= 0;

          if txDivisor = '1' then
```

```
      txData <= txBytesTransferred;
      mainSignal <= startingBit;
    else
      mainSignal <= idleSignal;
    end if;



  when startingBit =>
    txActivated <= '1';
    txSerialConnection <= '0';

    if newClock < clocksPerBits-1 then
      newClock <= newClock + 1;
      mainSignal   <= startingBit;
    else
      newClock <= 0;
      mainSignal   <= dataTransferred;
    end if;

  when dataTransferred =>
    txSerialConnection <= txData(bitIndexer);

    if newClock < clocksPerBits-1 then
      newClock <= newClock + 1;
      mainSignal   <= dataTransferred;
    else
      newClock <= 0;

      -- Check if we have sent out all bits
      if bitIndexer < 7 then
        bitIndexer <= bitIndexer + 1;
        mainSignal   <= dataTransferred;
      else
```

```
            bitIndexer <= 0;
            mainSignal   <= stopperBit;
          end if;
        end if;



      when stopperBit =>
        txSerialConnection <= '1';



        if newClock < clocksPerBits-1 then
          newClock <= newClock + 1;
          mainSignal   <= stopperBit;
        else
          txDone   <= '1';
          newClock <= 0;
          mainSignal   <= cleanAll;
        end if;

      when cleanAll =>
        txActivated <= '0';
        txDone   <= '1';
        mainSignal   <= idleSignal;



      when others =>
        mainSignal <= idleSignal;

    end case;
  end if;
end process;

txFinished <= txDone;
```

```
end Behavioral;
```

1G: Arduino Code for Color Sensors and RGB Leds

```
int aS0=22, aS1=23, aS2=24, aS3=25, aOUT=26;
int aRedReading, aBlueReading, aGreenReading, aWhiteReading;
int aRedPin=4, aGreenPin=3, aBluePin=2;

int bS0=27, bS1=28, bS2=29, bS3=30, bOUT=31;
int bRedReading, bBlueReading, bGreenReading, bWhiteReading;
int bRedPin=7, bGreenPin=6, bBluePin=5;

//int cS0=32, cS1=33, cS2=34, cS3=35, cOUT=36;
//int cRedReading, cBlueReading, cGreenReading, cWhiteReading;
//int cRedPin=10, cGreenPin=9, cBluePin=8;
//
//int dS0=37, dS1=38, dS2=39, dS3=40, dOUT=41;
//int dRedReading, dBlueReading, dGreenReading, dWhiteReading;
//int dRedPin=13, dGreenPin=12, dBluePin=11;

void setup() {
  pinMode(aS0, OUTPUT);
  pinMode(aS1, OUTPUT);
  pinMode(aS2, OUTPUT);
  pinMode(aS3, OUTPUT);
  pinMode(aOUT, INPUT);
  pinMode(aRedPin, OUTPUT);
  pinMode(aBluePin, OUTPUT);
  pinMode(aGreenPin, OUTPUT);
  digitalWrite(aS0, HIGH);
  digitalWrite(aS1, LOW);
```

```
  pinMode(bS0, OUTPUT);
  pinMode(bS1, OUTPUT);
  pinMode(bS2, OUTPUT);
  pinMode(bS3, OUTPUT);
  pinMode(bOUT, INPUT);
  pinMode(bRedPin, OUTPUT);
  pinMode(bBluePin, OUTPUT);
  pinMode(bGreenPin, OUTPUT);
  digitalWrite(bS0, HIGH);
  digitalWrite(bS1, LOW);

//  pinMode(cS0, OUTPUT);
//  pinMode(cS1, OUTPUT);
//  pinMode(cS2, OUTPUT);
//  pinMode(cS3, OUTPUT);
//  pinMode(cOUT, INPUT);
//  pinMode(cRedPin, OUTPUT);
//  pinMode(cBluePin, OUTPUT);
//  pinMode(cGreenPin, OUTPUT);
//  digitalWrite(cS0, HIGH);
//  digitalWrite(cS1, LOW);
//
//  pinMode(dS0, OUTPUT);
//  pinMode(dS1, OUTPUT);
//  pinMode(dS2, OUTPUT);
//  pinMode(dS3, OUTPUT);
//  pinMode(dOUT, INPUT);
//  pinMode(dRedPin, OUTPUT);
//  pinMode(dBluePin, OUTPUT);
//  pinMode(dGreenPin, OUTPUT);
//  digitalWrite(dS0, HIGH);
//  digitalWrite(dS1, LOW);
```

```
  Serial.begin(9600);
}

int detect_red(int num, int Sa, int Sb, int Sc){
  digitalWrite(Sa, LOW);
  digitalWrite(Sb, LOW);
  int red = pulseIn(Sc, LOW);
  if(num == 1)
    red = 255 - (red + 160);
  else if(num == 2)
    red = 255 - (red + 145);
  if(red < 0)
      red = 0;
  else if(red>255)
      red = 255;
  return red;
}
int detect_blue(int num, int Sa, int Sb, int Sc){
  digitalWrite(Sa, LOW);
  digitalWrite(Sb, HIGH);
  int blue = pulseIn(Sc, LOW);
  if(num == 1)
    blue = 255 - (blue + 170);
  else if(num == 2)
    blue = 255 - (blue + 95);
  if(blue < 0)
      blue = 0;
  else if(blue > 255)
      blue = 255;
  return blue;
}

int detect_green(int num, int Sa, int Sb, int Sc){
```

```
  digitalWrite(Sa, HIGH);
  digitalWrite(Sb, HIGH);
  int green = pulseIn(Sc, LOW);
  if(num == 1)
    green = 255 - (green + 150);
  else if(num == 2)
    green = 255 - (green + 80);
  if(green < 0)
      green = 0;
  else if(green > 255)
      green = 255;
  return green;
}

void loop() {
  aRedReading = detect_red(1, aS2, aS3, aOUT);
  aBlueReading = detect_blue(1, aS2, aS3, aOUT);
  aGreenReading = detect_green(1, aS2, aS3, aOUT);

    bRedReading = detect_red(2, bS2, bS3, bOUT);
    bBlueReading = detect_blue(2, bS2, bS3, bOUT);
    bGreenReading = detect_green(2, bS2, bS3, bOUT);

//    cRedReading = detect_red(cS2, cS3, cOUT);
//    cBlueReading = detect_blue(cS2, cS3, cOUT);
//    cGreenReading = detect_green(cS2, cS3, cOUT);
//
//    dRedReading = detect_red(dS2, dS3, dOUT);
//    dBlueReading = detect_blue(dS2, dS3, dOUT);
//    dGreenReading = detect_green(dS2, dS3, dOUT);

  setColors(aRedPin, aGreenPin, aBluePin, aRedReading,
aGreenReading, aBlueReading);
```

```
  setColors(bRedPin, bGreenPin, bBluePin, bRedReading,
bGreenReading, bBlueReading);
//  setColors(cRedPin, cGreenPin, cBluePin, cRedReading,
cGreenReading, cBlueReading);
//  setColors(dRedPin, dGreenPin, dBluePin, dRedReading,
dGreenReading, dBlueReading);


  delay(200);
}


void setColors(int pin1, int pin2, int pin3, int red, int green, int
blue)   {
  analogWrite(pin1 , 255-red);
  analogWrite(pin2 , 255-green);
  analogWrite(pin3 , 255-blue);
}




//  // detect brightness
//  digitalWrite(aS2, HIGH);
//  digitalWrite(aS3, LOW);
//  aWhiteReading = pulseIn(aOUT, LOW);
//
//  // Calibration of the colors' read values
//  aWhiteReading = 255 - aWhiteReading;
//
//  if(aWhiteReading < 0)
//    aWhiteReading = 0;
//
//  if(aWhiteReading > 255)
//    aWhiteReading = 255;



//  Serial.print("R: " + String(aRedReading));
```

```
//  Serial.print("\tG: " + String(aGreenReading));
//  Serial.print("\tB: " + String(aBlueReading));
//  Serial.print("\tW: " + String(aWhiteReading));
//  Serial.print("\r\n");
```

1H: 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]


##Buttons
set_property PACKAGE_PIN T18 [get_ports reset]
     set_property IOSTANDARD LVCMOS33 [get_ports reset]
set_property PACKAGE_PIN W19 [get_ports leftCommand]
     set_property IOSTANDARD LVCMOS33 [get_ports leftCommand]
set_property PACKAGE_PIN T17 [get_ports command]
     set_property IOSTANDARD LVCMOS33 [get_ports command]


##Pmod Header JA
#Sch name = JA1
```

```
set_property PACKAGE_PIN J1 [get_ports {rx}]
     set_property IOSTANDARD LVCMOS33 [get_ports {rx}]
#Sch name = JA2
set_property PACKAGE_PIN L2 [get_ports {tx}]
     set_property IOSTANDARD LVCMOS33 [get_ports {tx}]
#Sch name = JA3
set_property PACKAGE_PIN J2 [get_ports {pwmSignal1}]
     set_property IOSTANDARD LVCMOS33 [get_ports {pwmSignal1}]
#Sch name = JA4
set_property PACKAGE_PIN G2 [get_ports {pwmSignal2}]
     set_property IOSTANDARD LVCMOS33 [get_ports {pwmSignal2}]
#Sch name = JA7
set_property PACKAGE_PIN H1 [get_ports {pwmSignal3}]
     set_property IOSTANDARD LVCMOS33 [get_ports {pwmSignal3}]
#Sch name = JA8
set_property PACKAGE_PIN K2 [get_ports {pwmSignal4}]
     set_property IOSTANDARD LVCMOS33 [get_ports {pwmSignal4}]
##Sch name = JA9
#set_property PACKAGE_PIN H2 [get_ports {JA[6]}]
     #set_property IOSTANDARD LVCMOS33 [get_ports {JA[6]}]
##Sch name = JA10
#set_property PACKAGE_PIN G3 [get_ports {JA[7]}]
     #set_property IOSTANDARD LVCMOS33 [get_ports {JA[7]}]
```