DSD Project Report
Team 15

Chosen Project:
(B) Mini Garage

Team members:
1. Ahmed Gasser (55-25085) (T-6)
2. Amr Hegazy (55-0701) (T-7)
3. Ahmed sameh (55-13683) (T-24)
4. Omar Elharridy (55-0654) (T-6)
5. Seif Tarek Mostafa (55-24853) (T-6)
6. Zeyad Attia (55-11216) (T-21)

1.The Idea:-

An interactive "Mini Garage" project that uses an FPGA and an Arduino working in tandem. An Arduino listens for an approaching car using a force sensor, and an FPGA controls a servo motor that serves as the gate mechanism. The gate opens in response to a sensor signal, increasing the number of cars shown on a 7-segment display via the FPGA, and then closes. With the use of a force sensor for precise car detection, a servo motor for gate movement and a 7-segment display to display how many cars have entered the garage, this creative integration highlights the adaptability of FPGA and Arduino cooperation and guarantees an interesting and technically varied project experience.

2.Set of sensors/parts used:-

- Servo Motor (SG90 Micro Servo Motor) (✕1)
- Force Sensor (FSR 406) (✕1)
- Seven Segment Display (✕5)
- Arduino Uno R3 (✕1)

3.The Implementation:-

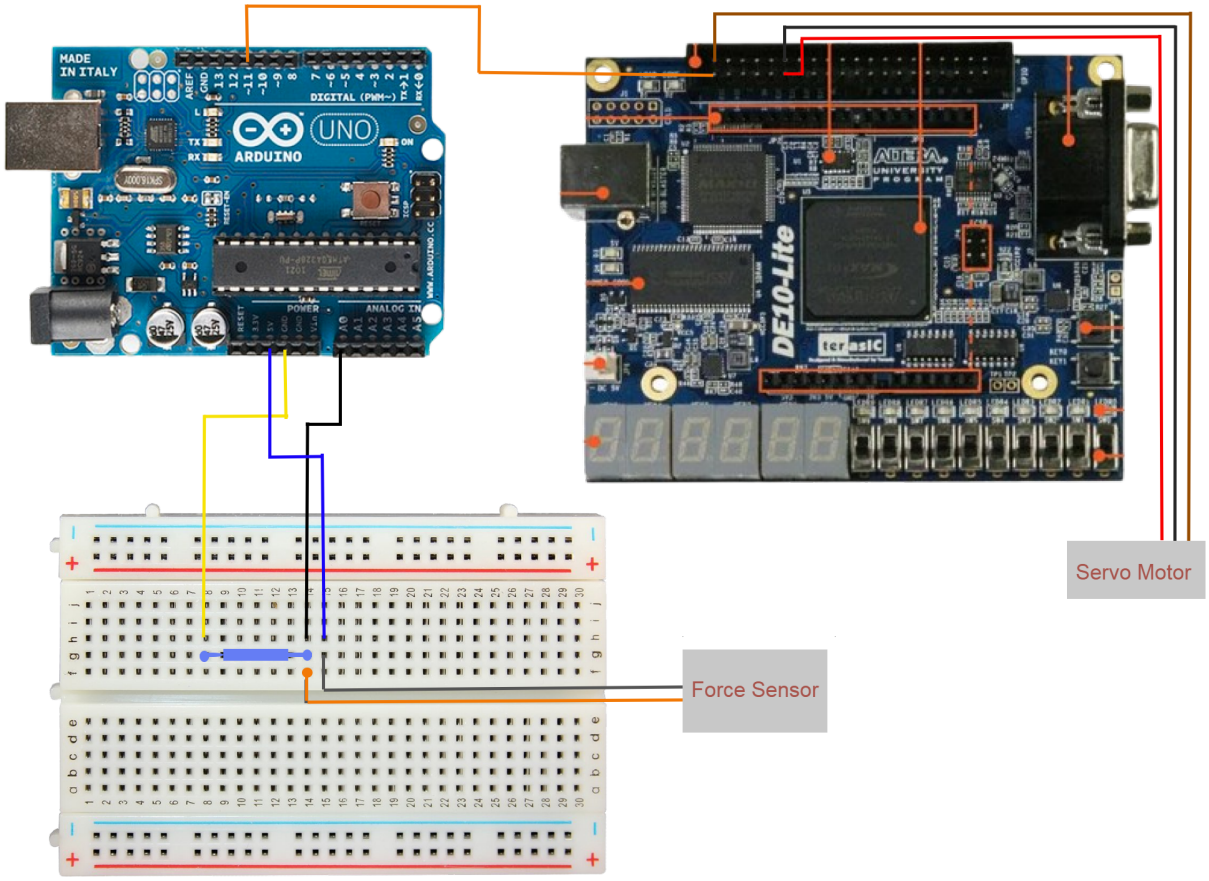
3.1. Pin Assignment:

digit2Out[2]	Output	PIN_B19	7	B7_N0	PIN_B19	2.5 V		12mA (default)	2 (default)	
digit2Out[1]	Output	PIN_A20	7	B7_N0	PIN_A20	2.5 V		12mA (default)	2 (default)	
digit2Out[0]	Output	PIN_B20	6	B6_N0	PIN_B20	2.5 V		12mA (default)	2 (default)	
digit3Out[6]	Output	PIN_E17	6	B6_N0	PIN_E17	2.5 V		12mA (default)	2 (default)	
digit3Out[5]	Output	PIN_D19	6	B6_N0	PIN_D19	2.5 V		12mA (default)	2 (default)	
digit3Out[4]	Output	PIN_C20	6	B6_N0	PIN_C20	2.5 V		12mA (default)	2 (default)	
digit3Out[3]	Output	PIN_C19	7	B7_N0	PIN_C19	2.5 V		12mA (default)	2 (default)	
digit3Out[2]	Output	PIN_E21	6	B6_N0	PIN_E21	2.5 V		12mA (default)	2 (default)	
digit3Out[1]	Output	PIN_E22	6	B6_N0	PIN_E22	2.5 V		12mA (default)	2 (default)	
digit3Out[0]	Output	PIN_F21	6	B6_N0	PIN_F21	2.5 V		12mA (default)	2 (default)	
digit4Out[6]	Output	PIN_F20	6	B6_N0	PIN_F20	2.5 V		12mA (default)	2 (default)	
digit4Out[5]	Output	PIN_F19	6	B6_N0	PIN_F19	2.5 V		12mA (default)	2 (default)	
digit4Out[4]	Output	PIN_H19	6	B6_N0	PIN_H19	2.5 V		12mA (default)	2 (default)	
digit4Out[3]	Output	PIN_J18	6	B6_N0	PIN_J18	2.5 V		12mA (default)	2 (default)	
digit4Out[2]	Output	PIN_E19	6	B6_N0	PIN_E19	2.5 V		12mA (default)	2 (default)	
digit4Out[1]	Output	PIN_E20	6	B6_N0	PIN_E20	2.5 V		12mA (default)	2 (default)	
digit4Out[0]	Output	PIN_F18	6	B6_N0	PIN_F18	2.5 V		12mA (default)	2 (default)	
f	Output	PIN_A8	7	B7_N0	PIN_A8	2.5 V		12mA (default)	2 (default)	
pwm	Output	PIN_W10	3	B3_N0	PIN_W10	2.5 V		12mA (default)	2 (default)	

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Slew Rate	Differential Pair	Strict Preservation
a	Input	PIN_V10	3	B3_N0	PIN_V10	2.5 V		12mA (default)			
clk	Input	PIN_P11	3	B3_N0	PIN_P11	2.5 V		12mA (default)			
digit0Out[6]	Output	PIN_C17	7	B7_N0	PIN_C17	2.5 V		12mA (default)	2 (default)		
digit0Out[5]	Output	PIN_D17	7	B7_N0	PIN_D17	2.5 V		12mA (default)	2 (default)		
digit0Out[4]	Output	PIN_E16	7	B7_N0	PIN_E16	2.5 V		12mA (default)	2 (default)		
digit0Out[3]	Output	PIN_C16	7	B7_N0	PIN_C16	2.5 V		12mA (default)	2 (default)		
digit0Out[2]	Output	PIN_C15	7	B7_N0	PIN_C15	2.5 V		12mA (default)	2 (default)		
digit0Out[1]	Output	PIN_E15	7	B7_N0	PIN_E15	2.5 V		12mA (default)	2 (default)		
digit0Out[0]	Output	PIN_C14	7	B7_N0	PIN_C14	2.5 V		12mA (default)	2 (default)		
digit1Out[6]	Output	PIN_B17	7	B7_N0	PIN_B17	2.5 V		12mA (default)	2 (default)		
digit1Out[5]	Output	PIN_A18	7	B7_N0	PIN_A18	2.5 V		12mA (default)	2 (default)		
digit1Out[4]	Output	PIN_A17	7	B7_N0	PIN_A17	2.5 V		12mA (default)	2 (default)		
digit1Out[3]	Output	PIN_B16	7	B7_N0	PIN_B16	2.5 V		12mA (default)	2 (default)		
digit1Out[2]	Output	PIN_E18	6	B6_N0	PIN_E18	2.5 V		12mA (default)	2 (default)		
digit1Out[1]	Output	PIN_D18	6	B6_N0	PIN_D18	2.5 V		12mA (default)	2 (default)		
digit1Out[0]	Output	PIN_C18	7	B7_N0	PIN_C18	2.5 V		12mA (default)	2 (default)		
digit2Out[6]	Output	PIN_B22	6	B6_N0	PIN_B22	2.5 V		12mA (default)	2 (default)		
digit2Out[5]	Output	PIN_C22	6	B6_N0	PIN_C22	2.5 V		12mA (default)	2 (default)		
digit2Out[4]	Output	PIN_B21	6	B6_N0	PIN_B21	2.5 V		12mA (default)	2 (default)		
digit2Out[3]	Output	PIN_A21	6	B6_N0	PIN_A21	2.5 V		12mA (default)	2 (default)		
digit2Out[2]	Output	PIN_B19	7	B7_N0	PIN_B19	2.5 V		12mA (default)	2 (default)		
digit2Out[1]	Output	PIN_A20	7	B7_N0	PIN_A20	2.5 V		12mA (default)	2 (default)		
digit2Out[0]	Output	PIN_B20	6	B6_N0	PIN_B20	2.5 V		12mA (default)	2 (default)		
digit3Out[6]	Output	PIN_E17	6	B6_N0	PIN_E17	2.5 V		12mA (default)	2 (default)		
digit3Out[5]	Output	PIN_D19	6	B6_N0	PIN_D19	2.5 V		12mA (default)	2 (default)		
digit3Out[4]	Output	PIN_C20	6	B6_N0	PIN_C20	2.5 V		12mA (default)	2 (default)		
digit3Out[3]	Output	PIN_C19	7	B7_N0	PIN_C19	2.5 V		12mA (default)	2 (default)		

In the Arduino Circuit, It operates using pin A0 to receive the analog signal , Then we use Pin 11 to transmit a signal to instruct the FPGA to activate the servo motor and open the gate of the garage.

3.2 Circuits



3.3. Code:

7 segment display code

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



Entity seven_segment is

    port(a: in std_logic_vector (3 downto 0);
   f :out std_logic_vector (6 downto 0)
    );

End seven_segment;

Architecture arch_seven_segment of seven_segment IS


begin

    WITH a SELECT

    f <= "1000000"  WHEN "0000",
    "1111001" WHEN "0001",
    "0100100" WHEN "0010",
    "0110000" WHEN "0011",
    "0011001" WHEN "0100",
    "0010010" WHEN "0101",
    "0000010" WHEN "0110",
    "1111000" WHEN "0111",
    "0000000" WHEN "1000",
    "0010000" WHEN "1001",
    "1111111" WHEN OTHERS;



end arch_seven_segment;

library ieee;
use ieee.std_logic_1164.all;


PACKAGE seven_segment_pkg IS

COMPONENT seven_segment
PORT(a: in std_logic_vector (3 downto 0);
   f :out std_logic_vector (6 downto 0)
    );

END COMPONENT;

END seven_segment_pkg;
```

7 segment display code Explanation:

This VHDL code defines a seven-segment display module that takes a 4-bit input `a` and outputs a 7-bit signal `f` corresponding to the segments needed to display the input digit in a seven-segment display. Inside the architecture block, a `WITH` statement is used in combination with a `SELECT` to map each possible input value of `a` to its respective output `f` value. For each input combination, a specific pattern of the seven-segment display is assigned. For instance, when the input is "0000," the output `f` is set to "1000000," which would display the digit 0 on a seven-segment display. The code also includes a package declaration, `seven_segment_pkg`, that contains the component declaration for the `seven_segment` entity, enabling its reuse in other parts of the design.

Motor Driver Code

```
LIBRARY ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;


entity motor_driver is

port( motorctrl ,clk: IN std_logic;
    pwm: OUT std_logic
);

end motor_driver;

architecture motor_driver_arch of motor_driver is
    signal cnt : INTEGER := 0;

    signal speedCtrl : INTEGER := 1070025;

    signal angleCtrl : INTEGER := 100000;
begin
    process(clk)
    begin
        if rising_edge(clk) then
            if cnt < speedCtrl then
                cnt <= cnt + 1;
            else
                cnt <= 0;

            end if;
        end if;
    end process;

     process(motorctrl)
     begin
        if motorctrl = '1' THEN
            angleCtrl <= 50000;
        else
            angleCtrl <= 100000;

        end if;
    end process;

    pwm <= '1' when cnt < angleCtrl else '0';
end motor_driver_arch;

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

PACKAGE motor_driver_pkg IS

    COMPONENT motor_driver
    port( motorctrl ,clk: IN std_logic;
        pwm: OUT std_logic
    );

    END COMPONENT;

END motor_driver_pkg;
```

Motor Driver Code Explanation:

This VHDL code defines a motor driver module responsible for generating a PWM (Pulse Width Modulation) signal based on control inputs. It contains an `entity` named `motor_driver` with input ports for `motorctrl` and `clk`, and an output port `pwm`. Inside the `architecture` block, two processes are implemented. The first process, triggered by the `clk`, generates a counter (`cnt`) that increments until it reaches a value specified by `speedCtrl`, then resets. The second process monitors the `motorctrl` input and adjusts the `angleCtrl` value accordingly. Finally, the PWM output `pwm` is set based on whether the `cnt` is less than `angleCtrl`, providing a variable duty cycle based on these control signals. Additionally, the code includes a package `motor_driver_pkg` containing a component declaration for the `motor_driver` entity, enabling its use in other parts of the design.

Main code

```vhdl
LIBRARY IEEE;

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

use work.seven_segment_pkg.all;
use work.motor_driver_pkg.all;



entity DSD is
  port (
    a, clk: in std_logic;
     digit0Out, digit1Out, digit2Out, digit3Out, digit4Out :out std_logic_vector (6 downto 0);

     f, pwm: out std_logic
  );
end entity DSD;


architecture MyArchitecture of DSD is


  signal carCountSignal : INTEGER := 1;


  signal digit0Bin : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0000";
  signal digit1Bin : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0000";
  signal digit2Bin : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0000";
  signal digit3Bin : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0000";
  signal digit4Bin : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0000";

begin

   f <= a;



    digit0: seven_segment PORT MAP (digit0Bin, digit0Out);
    digit1: seven_segment PORT MAP (digit1Bin, digit1Out);
    digit2: seven_segment PORT MAP (digit2Bin, digit2Out);
    digit3: seven_segment PORT MAP (digit3Bin, digit3Out);
    digit4: seven_segment PORT MAP (digit4Bin, digit4Out);

    motorStage: motor_driver PORT MAP (a,clk,pwm);



    process(a) is
    begin

    IF a = '1' THEN
        carCountSignal <= carCountSignal + 1;

        digit0Bin <= std_logic_vector(to_unsigned((carCountSignal mod 10), digit0Bin'length));

        digit1Bin <= std_logic_vector(to_unsigned((carCountSignal / 10 mod 10), digit1Bin'length));

        digit2Bin <= std_logic_vector(to_unsigned((carCountSignal / 100 mod 10), digit2Bin'length));

        digit3Bin <= std_logic_vector(to_unsigned((carCountSignal / 1000 mod 10), digit3Bin'length));

        digit4Bin <= std_logic_vector(to_unsigned((carCountSignal / 10000 mod 10), digit4Bin'length));

    ELSE END IF;




    end process;




end architecture MyArchitecture;
```

Main code Explanation:

This VHDL code defines an FPGA-based system, `DSD`, designed to interact with an Arduino and a Force-Sensing Resistor (FSR). It operates a gate, represented by a servo motor, and counts the number of passing cars. The `DSD` entity contains inputs for signals from the Arduino (`a`) and a clock (`clk`), as well as outputs for displaying individual digit outputs on seven-segment displays (`digit0Out` to `digit4Out`), a combined output `f`, and a PWM output `pwm` for motor control. Within its architecture, the code initializes counting and digit representation signals. On detection of a car passage (`a` signal high), the count increments (`carCountSignal`) and translates into binary representations for each digit. These binary values are assigned to signals controlling seven-segment displays for visualizing the car count. Additionally, the code links the seven-segment display modules and the motor driver components to the respective physical devices, enabling the display of the count and gate control based on the input signals received from the Arduino and FSR.

Arduino Code Bonus

```
/* FSR testing sketch.

Connect one end of FSR to 5V, the other end to Analog 0.
Then connect one end of a 10K resistor from Analog 0 to ground
Connect LED from pin 11 through a resistor to ground

For more information see www.ladyada.net/learn/sensors/fsr.html */

int fsrAnalogPin = 0; // FSR is connected to analog 0
int gatePin = 11;      // connect Red LED to pin 11 (PWM pin)
int fsrReading;      // the analog reading from the FSR resistor divider
bool carPresent = 0;


void setup(void) {
  Serial.begin(9600);   // We'll send debugging information via the Serial monitor
  pinMode(gatePin, OUTPUT);
}

void loop(void) {
  fsrReading = analogRead(fsrAnalogPin);
  Serial.print("Analog reading = ");
  Serial.println(fsrReading);

  if (fsrReading > 50 && carPresent){


  }else if (fsrReading > 50 && !carPresent){

    digitalWrite(gatePin,HIGH);
    carPresent=1;

  }else{
    delay(1000);
    digitalWrite(gatePin,LOW);
    carPresent=0;
  }

  delay(100);
}
```

Arduino Code Bonus Explanation

This Arduino code is a sketch for a Force-Sensing Resistor (FSR). It sets up the connections for the FSR by linking one end to 5V and the other to Analog pin 0, with a 10K resistor bridging Analog 0 to ground. Additionally, an LED is connected to pin 11 through a resistor to indicate the gate's status. The sketch initializes variables such as `fsrAnalogPin` for the FSR's analog input, `gatePin` for the LED indicator, `fsrReading` to hold analog readings from the FSR, and `carPresent` as a boolean flag to track car presence. In the `setup()` function, it initiates serial communication for debugging and sets `gatePin` as an output. In the `loop()` function, it continuously reads the analog value from the FSR. If the FSR reading exceeds a threshold value of 50 and a car is already present, it remains unchanged. If the reading exceeds 50 and no car is detected, it signals the gate (LED at `gatePin`) to open by setting it `HIGH` and updates the `carPresent` flag. Otherwise, if no significant pressure is detected, it delays for a second, closes the gate by setting `gatePin` to `LOW`, and resets the `carPresent` flag. The delay at the end of the loop ensures a periodic check of the FSR reading.

Result:

We finally and successfully got everything running. Let me give you an overview of the full project and how it all runs together to create the automated mini garage that keeps track of the number of cars that enter it.
We have the gate which is powered by a motor and the weight sensor which is used to sense the car approaching the gate both connected through an Arduino, where the Arduino checks the weight sensor approximately every second until a “car” weight is applied after which he orders the motor to open the gate and stay open for about 2 seconds giving the “car” plenty of time to pass.
Simultaneously as the car enters the garage the FPGA count is incremented by 1 indicating the passing of a car. We implemented 5 of the 7 segments of the FPGA making our mini garage hold up to 99999 cars.
Finally, a picture of our mini garage