Florida International University

EEL4740 – Embedded Systems

Zybo Z7 Obstacle Avoidance Vehicle

Authors

Name: Dyanette Arroyo	PID: [redacted]
Name: Jonathan Baskharoun	PID: [redacted]
Name: Jose Hernandez	PID: [redacted]
Name: Alejandro Perez	PID: [redacted]

Project Objectives:

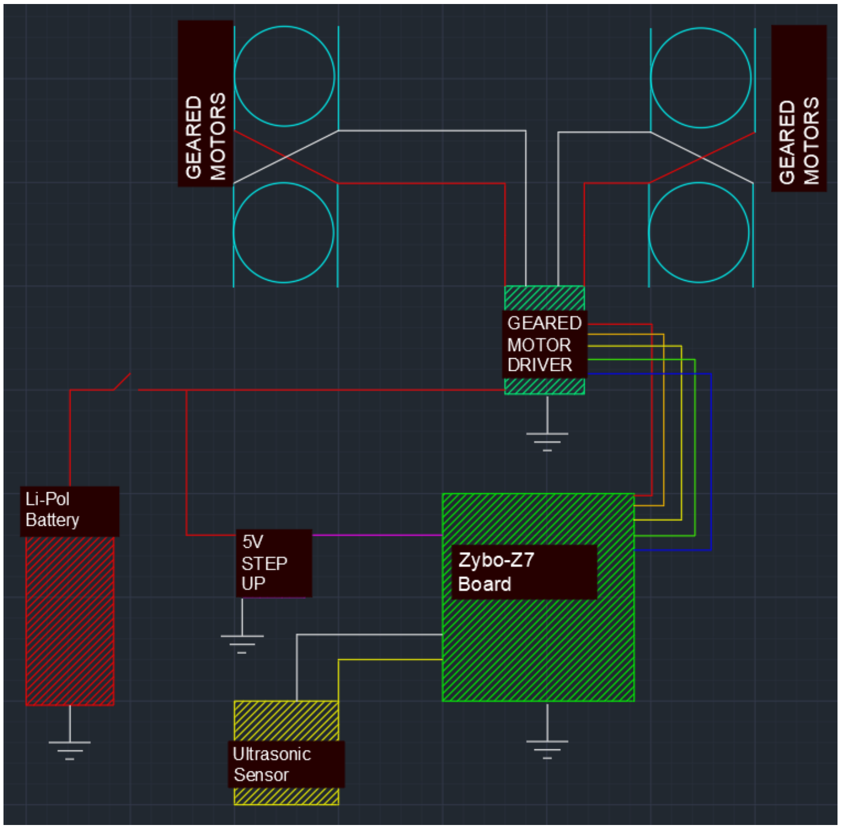
- Build a robotic vehicle which is able to autonomously avoid obstacles using an ultrasonic sensor and inbuilt motors
  - Design a clock delay function
  - Write functions to get distance using an ultrasonic sensor
  - Write functions to perform robot actuation using a motor
- Build hardware and electrical system for vehicle
  - Design and manufacture mechanical parts of the robot using a 3D printer
  - Design and implement power system

Components used

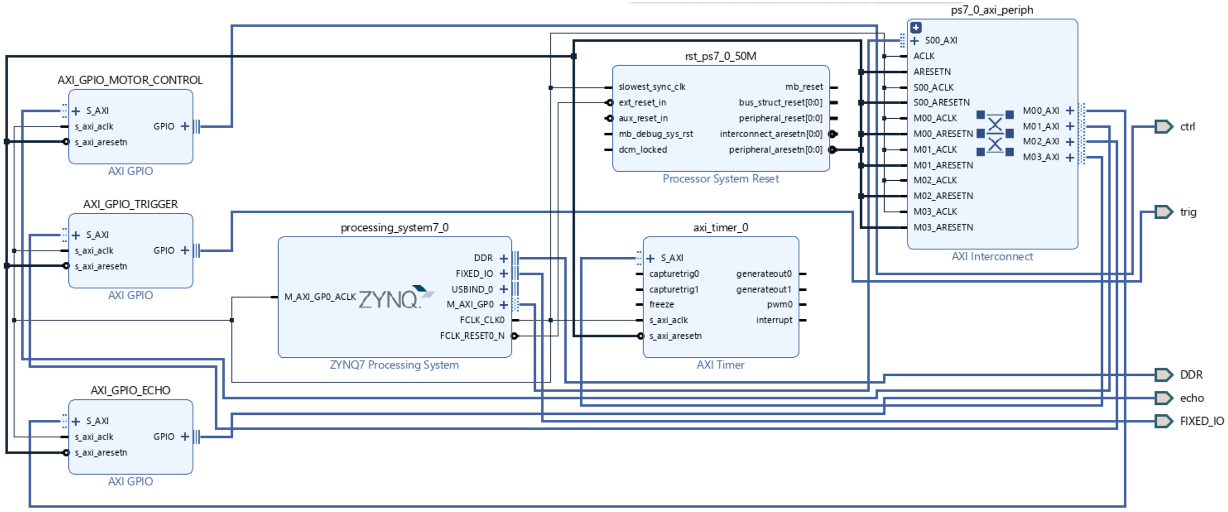
- Zybo-Z7 7020
- Motor Driver DRV8834
- Wires
- 5V Li-Pol Power Supply
- Geared motors
- 3D printing filament (PETG)
- Screws and nuts
- Plastic Wheels
- Ultrasonic sensor

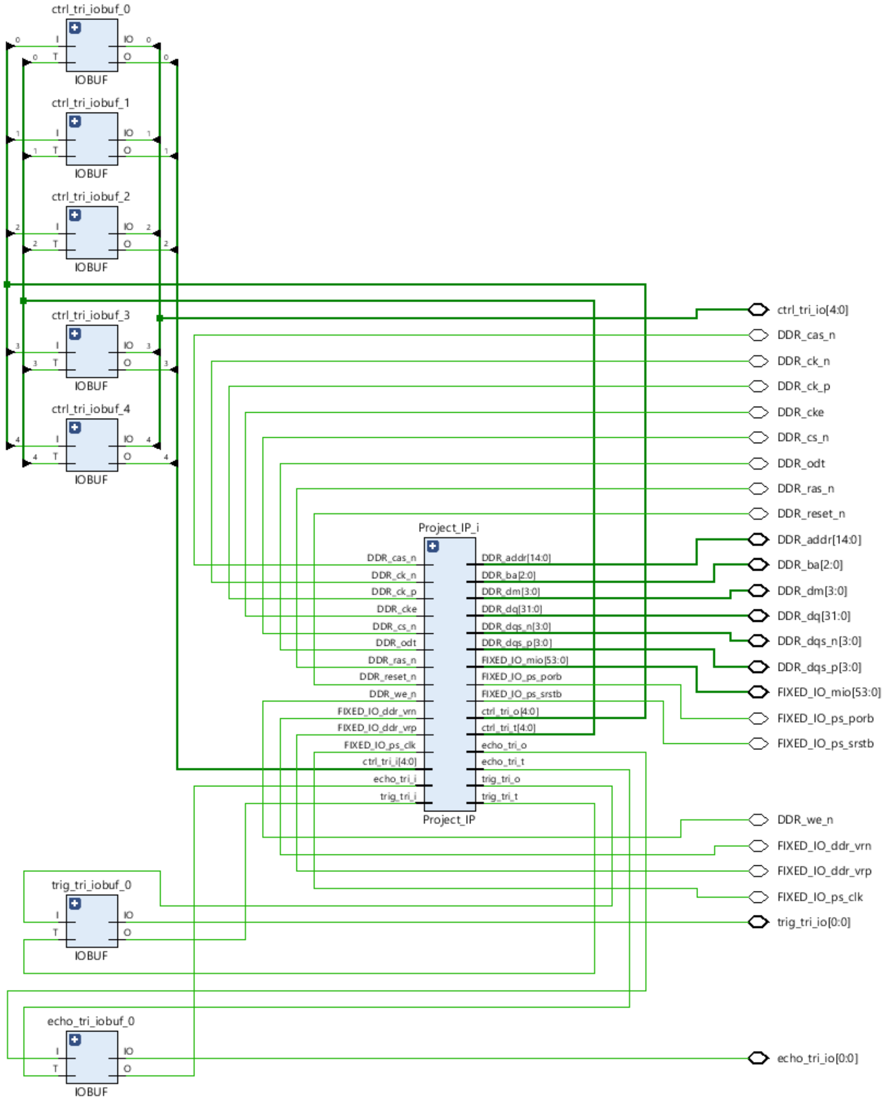
Picture or Schematic of the System

Hardware Schematic

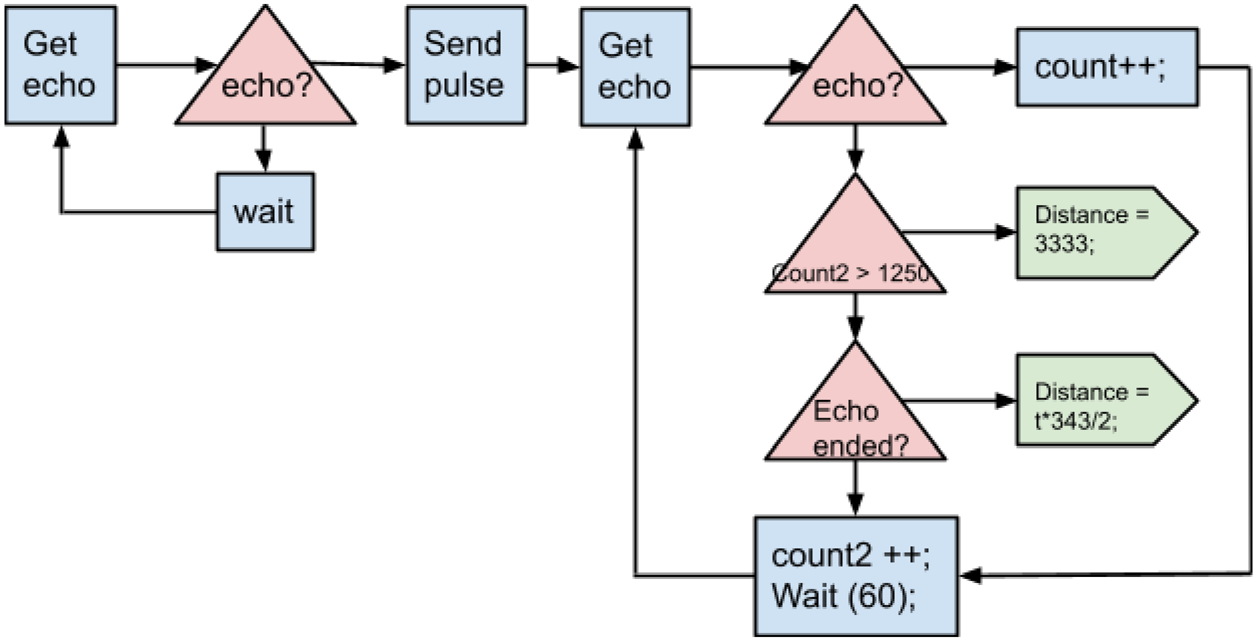


Hardware Design (Vivado)

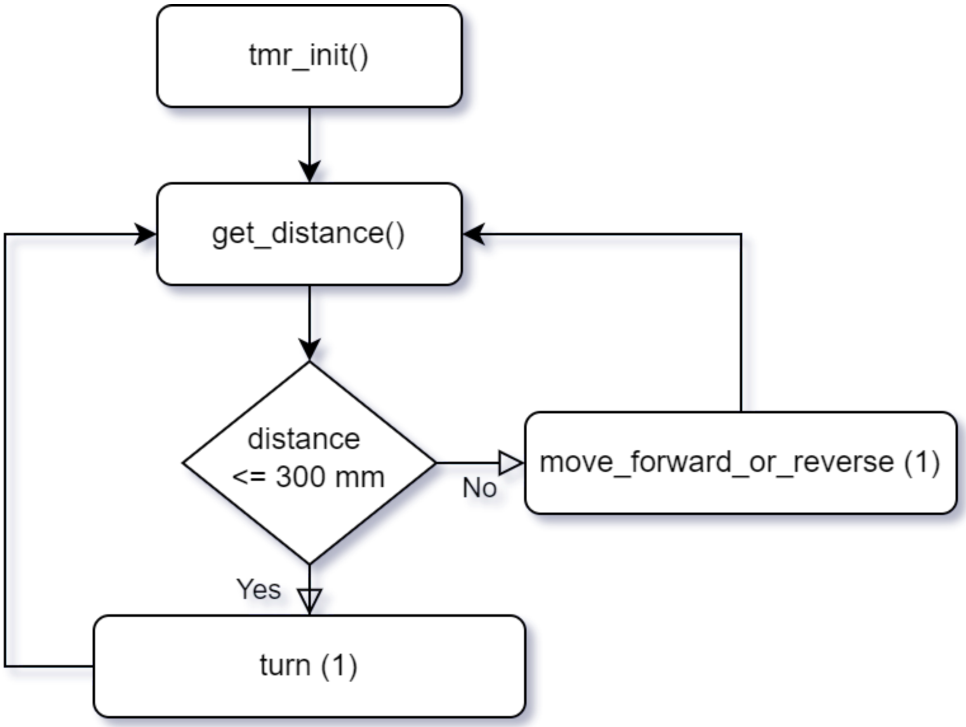



ctrl_tri_iobuf_0
IOBUF
ctrl_tri_iobuf_1
IOBUF
ctrl_tri_iobuf_2
IOBUF
ctrl_tri_iobuf_3
IOBUF
ctrl_tri_iobuf_4
IOBUF
I
T
IO
O
ctrl_tri_io[4:0]
DDR_cas_n
DDR_ck_n
DDR_ck_p
DDR_cke
DDR_cs_n
DDR_odt
DDR_ras_n
DDR_reset_n
Project_IP_i
DDR_cas_n
DDR_ck_n
DDR_ck_p
DDR_cke
DDR_cs_n
DDR_odt
DDR_ras_n
DDR_reset_n
DDR_we_n
FIXED_IO_ddr_vrn
FIXED_IO_ddr_vrp
FIXED_IO_ps_clk
ctrl_tri_i[4:0]
echo_tri_i
trig_tri_i
DDR_addr[14:0]
DDR_ba[2:0]
DDR_dm[3:0]
DDR_dq[31:0]
DDR_dqs_n[3:0]
DDR_dqs_p[3:0]
FIXED_IO_mio[53:0]
FIXED_IO_ps_porb
FIXED_IO_ps_srstb
ctrl_tri_o[4:0]
ctrl_tri_t[4:0]
echo_tri_o
echo_tri_t
trig_tri_o
trig_tri_t
Project_IP
DDR_addr[14:0]
DDR_ba[2:0]
DDR_dm[3:0]
DDR_dq[31:0]
DDR_dqs_n[3:0]
DDR_dqs_p[3:0]
FIXED_IO_mio[53:0]
FIXED_IO_ps_porb
FIXED_IO_ps_srstb
DDR_we_n
FIXED_IO_ddr_vrn
FIXED_IO_ddr_vrp
FIXED_IO_ps_clk
trig_tri_io[0:0]
trig_tri_iobuf_0
IOBUF
echo_tri_iobuf_0
IOBUF
echo_tri_io[0:0]

Ultrasonic sensor flowchart (Vitis)



Main function flowchart (Vitis)



VHDL Code

```
#include <stdio.h>
#include "platform.h"
#include "xtmrctr.h"
#include "xparameters.h"
#include "xil_printf.h"
#include "xgpio.h"
#include "xil_types.h"
#include "time.h"

// Get device IDs from xparameters.h
#define CTRL_ID XPAR_AXI_GPIO_MOTOR_CONTROL_DEVICE_ID
#define TRIG_ID XPAR_AXI_GPIO_TRIGGER_DEVICE_ID
#define ECHO_ID XPAR_AXI_GPIO_ECHO_DEVICE_ID
#define CTRL_CHANNEL 1
#define TRIG_CHANNEL 1
#define ECHO_CHANNEL 1

// Timer initialization function

XTmrCtr tmr;
void tmr_init(){
        int status =  XTmrCtr_Initialize(&tmr, XPAR_AXI_TIMER_0_DEVICE_ID);

        //if(status == XST_SUCCESS)
                //xil_printf("TMR INIT SUCCESSFUL\n");
        //else
                //xil_printf("TMR INIT FAILED\n");

        status = XTmrCtr_SelfTest(&tmr, 0);

        //if(status == XST_SUCCESS)
                //xil_printf("TMR SELFTEST SUCCESSFUL\n");
        //else
                //xil_printf("TMR SELFTEST FAILED\n");
}

void wait(int cycles){

        u32 wait = cycles;
        init_platform();

        XTmrCtr_Stop(&tmr, 0);
        XTmrCtr_SetResetValue(&tmr, 0, wait);
```

```
    XTmrCtr_Reset(&tmr, 0);

    u32 option = XTmrCtr_GetOptions(&tmr, 0);
    XTmrCtr_SetOptions(&tmr, 0, option | XTC_DOWN_COUNT_OPTION );

    XTmrCtr_Start(&tmr, 0);
    while(!XTmrCtr_IsExpired(&tmr, 0));
}

void send_trig_pulse(){

    XGpio_Config *cfg_ptr;
    XGpio trig_device;

    // Initialize Trigger Device
    cfg_ptr = XGpio_LookupConfig(TRIG_ID);
    XGpio_CfgInitialize(&trig_device, cfg_ptr, cfg_ptr->BaseAddress);

    // Set Trigger Tristate (Output)
    XGpio_SetDataDirection(&trig_device, TRIG_CHANNEL, 0);

    //Sending 10 us pulse
    XGpio_DiscreteWrite(&trig_device, TRIG_CHANNEL, 1);
    wait(60);
    XGpio_DiscreteWrite(&trig_device, TRIG_CHANNEL, 0);

    //xil_printf("Pulse sent\n");

    cleanup_platform();
}

int get_distance(){

    XGpio_Config *cfg_ptr;
    XGpio echo_device;
    u32 echo_state;
    int count = 0;
    int count2 = 0;
    u32 echo_pulse_started = 0;
    int distance = 333;

    // Initialize Echo Device
    cfg_ptr = XGpio_LookupConfig(ECHO_ID);
    XGpio_CfgInitialize(&echo_device, cfg_ptr, cfg_ptr->BaseAddress);
    // Set Echo Tristate (Input)
    XGpio_SetDataDirection(&echo_device, TRIG_CHANNEL, 1);

    while (1) {
        //xil_printf("Stuck in loop 1");
        echo_state = XGpio_DiscreteRead(&echo_device, ECHO_CHANNEL);
        if (echo_state){
            wait(5000000);
        }
        else {
            send_trig_pulse();
            wait(60);
            while(1) {
                //xil_printf("Stuck in loop 2");
                echo_state = XGpio_DiscreteRead(&echo_device,
ECHO_CHANNEL);
                //xil_printf("echo_state: %d\n", echo_state);

                if (echo_state) {
                    echo_pulse_started = 1;
                    count += 1;
                    //xil_printf("count: %d\n", count);
                }
                if (count2 >= 1250) {
                    distance = 3333;
                    break;
                }
                if ((echo_pulse_started) && (!echo_state)) {

                    distance = (((count) * 343)/2/100); // in mm
                    //xil_printf("%d\n", (int)(two));
                    break;
                }
                count2 += 1;
                wait(60);
                //xil_printf("count: %d\n", distance);
            }
        }
        //xil_printf("Broke out");
        break;
    }
    return distance;
}

void move_forward_or_reverse(unsigned char forward_or_reverse){
    XGpio_Config *cfg_ptr;
    XGpio ctrl_device;

    // Initialize Control Device
    cfg_ptr = XGpio_LookupConfig(CTRL_ID);
    XGpio_CfgInitialize(&ctrl_device, cfg_ptr, cfg_ptr->BaseAddress);

    // Set Control Tristate (Output)
    XGpio_SetDataDirection(&ctrl_device, CTRL_CHANNEL, 0);

    if (forward_or_reverse){
        XGpio_DiscreteWrite(&ctrl_device, CTRL_CHANNEL, 0b01111);
    }
    else{
        XGpio_DiscreteWrite(&ctrl_device, CTRL_CHANNEL, 0b10111);
    }
}

void turn(unsigned char right_or_left){
    XGpio_Config *cfg_ptr;
    XGpio ctrl_device;

    // Initialize Control Device
    cfg_ptr = XGpio_LookupConfig(CTRL_ID);
    XGpio_CfgInitialize(&ctrl_device, cfg_ptr, cfg_ptr->BaseAddress);

    // Set Control Tristate (Output)
    XGpio_SetDataDirection(&ctrl_device, CTRL_CHANNEL, 0);

    if (right_or_left){
        XGpio_DiscreteWrite(&ctrl_device, CTRL_CHANNEL, 0b00111);
    }
    else{
        XGpio_DiscreteWrite(&ctrl_device, CTRL_CHANNEL, 0b11111);
    }
}

void stop(unsigned char stop){
    XGpio_Config *cfg_ptr;
    XGpio ctrl_device;

    // Initialize Control Device
    cfg_ptr = XGpio_LookupConfig(CTRL_ID);
    XGpio_CfgInitialize(&ctrl_device, cfg_ptr, cfg_ptr->BaseAddress);

    // Set Control Tristate (Output)
    XGpio_SetDataDirection(&ctrl_device, CTRL_CHANNEL, 0);
    if (stop){
        XGpio_DiscreteWrite(&ctrl_device, CTRL_CHANNEL, 0b00000);
    }
}

int main() {
    int distance_measured;
    tmr_init();

    while (1) {
        //send_trig_pulse();
        distance_measured = get_distance();
        //xil_printf("Distance: %d\n", distance_measured);
        if (distance_measured <= 300){
            turn(1);
            wait(100000000);
        }
        else {
            move_forward_or_reverse(1);
        }
        //wait(100000000);//~2 second delay
    }
    return 0;
}
```

Troubleshooting

- Had unexpected behavior when reading the distance from the ultrasonic sensor because of integer division truncating the distance counter to 0. Issue could be fixed by multiplying the number received by a large power of 10 before doing further calculations. Working with inflated integers is more favorable than trying to implement floats on hardware.

- The motor's motion only moves forward or turns right, and we cannot get the motor to make the vehicle turn left. Error may be with pin configuration, or the current / power being transmitted to the motors. Further testing can be done to show where the issue is coming from, but the motor is still usable in its current state.

- The tmr_init() function was being called repeatedly from an infinite runtime loop causing a large number of print statements to fill the serial monitor.

Instead of calling the init function from the wait() function, moved it to the beginning of main() to be run once.

Recommendations

- Use more sensors to avoid a wider range of obstacles
- Clean up the code used to run the motors to include less of the commented code use for debugging and analyzing behaviors
- Refactor some of the branching and looping code to use less indentation and be more clear / readable
- Improve the flagging and activating code so that the motor and vehicle have full 4 degrees of freedom (forward, reverse, left, and right)
- Re-do the project using a Raspberry Pi with Arduino code

Conclusions

We see now more often than not autonomous systems in our daily lives, from self-driving cars to robotics vacuum cleaners. Human error is a growing issue in our roads, developing technologies in automation are reducing risks of crashes and driver errors significantly. In this project we simulated a basic autonomous vehicle continuously avoiding obstacles with one ultrasonic sensor in a FPGA board. With further modifications and more sensing capabilities these designs are key to increasing safety features in our everyday living. Results show that FPGA boards are a great tool for prototyping autonomous robots.The robot was successful at avoiding obstacles, however, it needs further sensors to catch obstacles in outer angles. Although the project was successful, for a basic one sensor robot there are easier resources to use for hardware and programming. Using a FPGA board would be expensive and time consuming to learn to program. Working with open source boards like Arduino would result in the same outcome while also being cost-effective and easy to use.

Gallery