ECE 520

System on Chip Design

Final Project Report

FPGA/SoC Robot Car: Zedbot

Performed By: Aaron Joseph Nanas

May 17, 2021

Spring 2021

Table of Contents

Section	Pg.
Overview	2
Components Used	2
Interfacing MIO Pmod	2
Results	3-10
References	11
Appendix	12-21

Overview:

This project aims to construct a simple robot car with a pan-and-tilt arm (2-axis) and a gripper using the SoC design flow. The project is implemented using a Zedboard and Digilent Pmods. Vivado 2019.2 with Vitis was used. The design goals are as follows:

- Using ServoCity's open-source 3D models, design a 3D model that can properly house four DC motors with encoders and one servo motor
- The design will contain an upper-level platform for a simple 2-DoF (Degrees of Freedom) arm that will eventually hold a camera
- The structure must be mostly 3D printed
- Two joysticks will control the motion for the DC motors and the servo motor

The overall structure is mostly 3D printed with Prusament PLA filament, excluding standoffs, nuts, and bolts. The 3D design was created in FreeCAD using ServoCity's 3D models. Moreover, the gripper model was obtained from an open-source Thingiverse project.

Components Used:

The Digilent Pmods used are the following:

- Two Pmod JSTK for motion control
- Two Pmod DHB1 to drive four DC motors with encoders. These motors control the wheels and the 2-DoF arm
- One Pmod CON3 to interface the servo
- The Pmod CON3 is connected to the Zedboard's MIO Pmod (JE)
    - This Pmod also allows the servo to be separately powered
        - An external power supply (6V) was used to power both the DC motors and the servo motor

Interfacing MIO Pmod:

The four Pmod ports (JA, JB, JC, JD) of the Zedboard were already used by the two Pmod JSTK2 and the two Pmod DHB1. In order to interface the servo motor, the MIO Pmod, specifically the JE1 port, was used to create a simple PWM output. This can easily be done by writing directly to the pins and using the usleep() function to generate both the frequency required by the servo motor and the duty cycle needed to make a rotation. The driver functions needed for the PS GPIO can be found in the "XGpiops.h" header file. The function that implements a simple PWM output is shown in Figure 2. Moreover, the pins can be found in the Zedboard's Hardware User Guide, as shown in Figure 3.

The rest of the code can be found in the Appendix section of the report, which will show the functions to drive the car and the push button ISR.

Results:

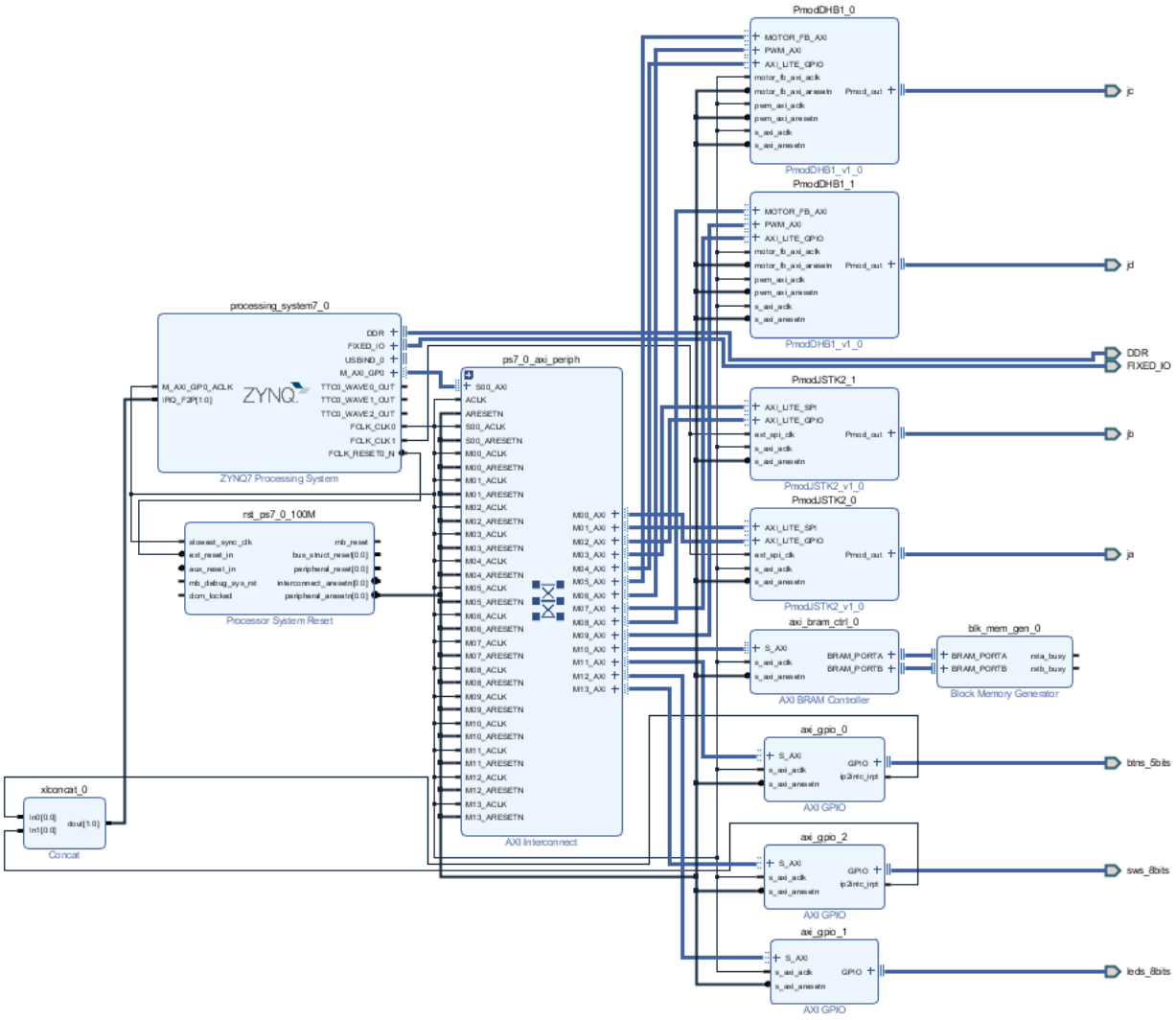

Fig. 1: Zedbot Block Diagram

```
int gripperControl() {
	int Status;

	ConfigPtr = XGpioPs_LookupConfig(XPAR_XGPIOPS_0_DEVICE_ID);
	Status = XGpioPs_CfgInitialize(&GpioPs, ConfigPtr,
	ConfigPtr->BaseAddr);

	if (Status != XST_SUCCESS) {
		return XST_FAILURE;
	}

	// Set directions and output enable for PMOD JE1 Pin
	XGpioPs_SetDirectionPin(&GpioPs, JE1_MIO, 1);
	XGpioPs_SetOutputEnablePin(&GpioPs, JE1_MIO, 1);

	if (rawdata1.Trigger == 1) {
		XGpioPs_WritePin(&GpioPs, JE1_MIO,1);
		usleep(1000); // Set output high for 1 ms (0 degrees)
		XGpioPs_WritePin(&GpioPs, JE1_MIO,0);
		usleep(20000); // Set output low for the rest of the period
	}

	else if (rawdata2.Trigger == 1) {
		XGpioPs_WritePin(&GpioPs, JE1_MIO, 1);
		usleep(2000); // Set output high for 2 ms (180 degrees)
		XGpioPs_WritePin(&GpioPs, JE1_MIO, 0);
		usleep(20000); // Set output low for the rest of the period
	}

	else {
		XGpioPs_WritePin(&GpioPs, JE1_MIO, 0);
		usleep(1000);
	}

	return XST_SUCCESS;
}
```

Fig. 2: gripperControl() function to generate a PWM output

Pmod	Signal Name	Zynq pin	MIO
JE1 MIO Pmod	JE1	A6	MIO13
	JE2	G7	MIO10
	JE3	B4	MIO11
	JE4	C5	MIO12
	JE7	G6	MIO0
	JE8	C4	MIO9
	JE9	B6	MIO14
	JE10	E6	MIO15

Fig. 3: Pins for Zedboard's JE MIO Pmod

Figure 4 presents the 3D two-axis arm model that I designed using ServoCity's open-source files.

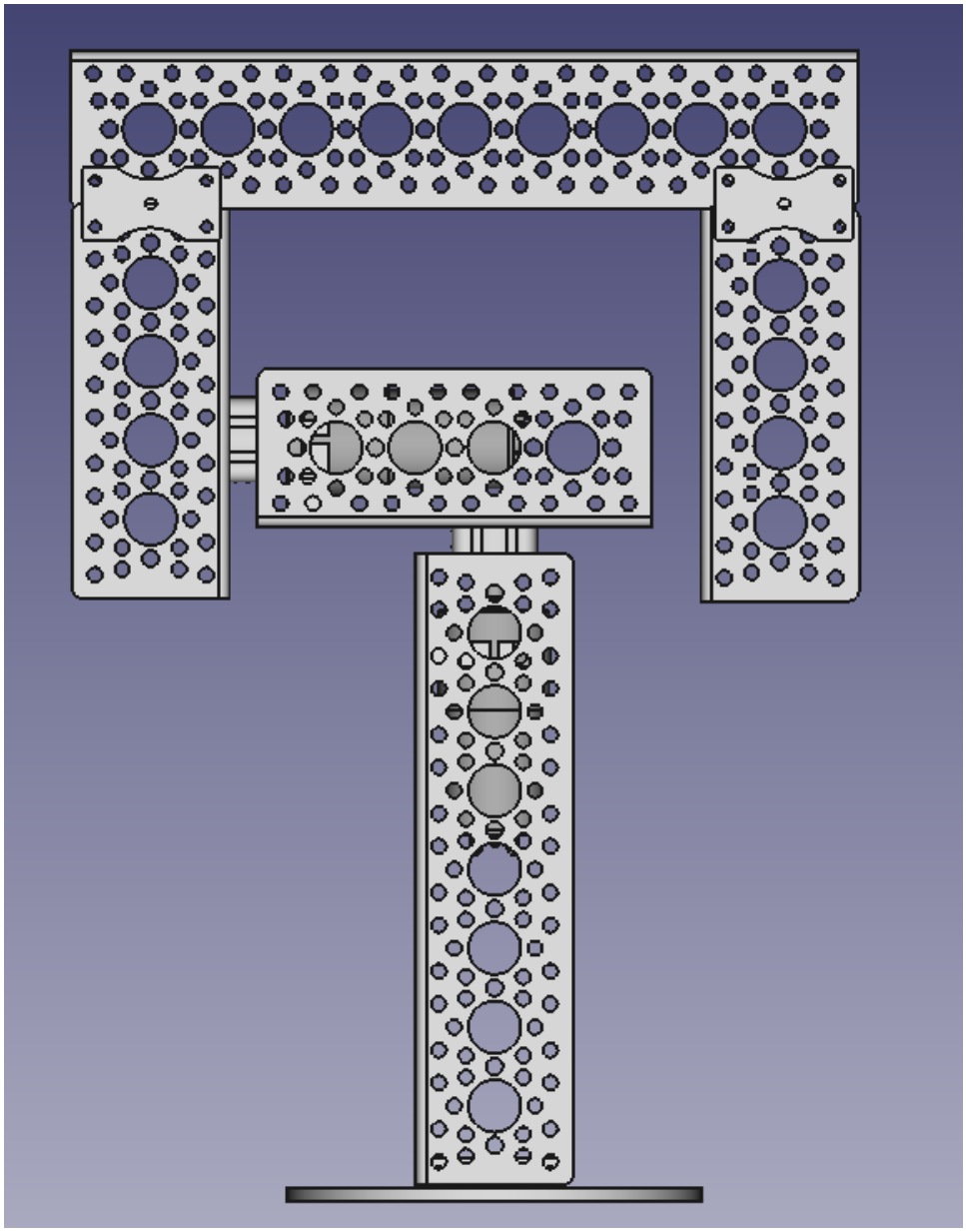

Fig. 4: Simple pan-and-tilt arm

Since this is the project's initial design, the Zedbot can only perform simple tasks:

- It can drive forward and backward. It currently does not have steering capability
- It can fully rotate the base of the arm, or move it up or down
    - The idea is that in the future, a camera will be implemented with the Zedboard, providing motion control to a camera and control its positioning
- It can grab something close to it with a gripper
- Motor speed can be configured with the push button interrupt service routine (ISR)

The following lists the functions used in the main program, with short descriptions for each function:

- void initializePMOD()
    - This initializes the Pmod JSTK2 and the Pmod DHB1
- void runRobotArm()
    - This initializes the default speed of the motors
    - It makes a call to the push button ISR, which can configure the motor speeds
    - It calls the other functions to drive the robot. It also continuously reads the X and Y values of the two joysticks, and then it initializes the RGB LED located on the joystick
- void rotateArm()
    - This will either tilt the two-axis arm forward or backward, depending on the first joystick's X and Y values
- void rotateBase()
    - This will rotate the base of the arm either counter-clockwise or clockwise, depending on the first joystick's X and Y values
- void driveForward()
    - This will allow the car to move forward until the specified amount of positive edges is reached (via the feedbackCounter function) depending on the second joystick's Y values
- void driveBackward()
    - This will allow the car to move backward until the specified amount of positive edges is reached (via the feedbackCounter function) depending on the second joystick's X values
- int gripperControl()
    - This controls the gripper using the Pmod MIO JE1 port. It uses the usleep() function to generate the frequency and the duty cycle. It directly writes to the PS GPIO using the XGpiops.h header file

- void feedbackCounter_1() and void feedbackCounter_2()
    - These functions enable the Pmod DHB1 and it will calculate the amount of positive edges when the motor runs. Note that the DC motor has encoders
- void BTN_Intr_Handler
    - The push button interrupt service routine that will configure the motor speeds. Pressing the center button will display the current speeds
- int InterruptSystemSetup
    - Enables the interrupt system for the push buttons
- int IntcInitFunction()
    - This initializes the interrupt controller, and it will enable the GPIO interrupts

The fully assembled car is shown in the following figure:

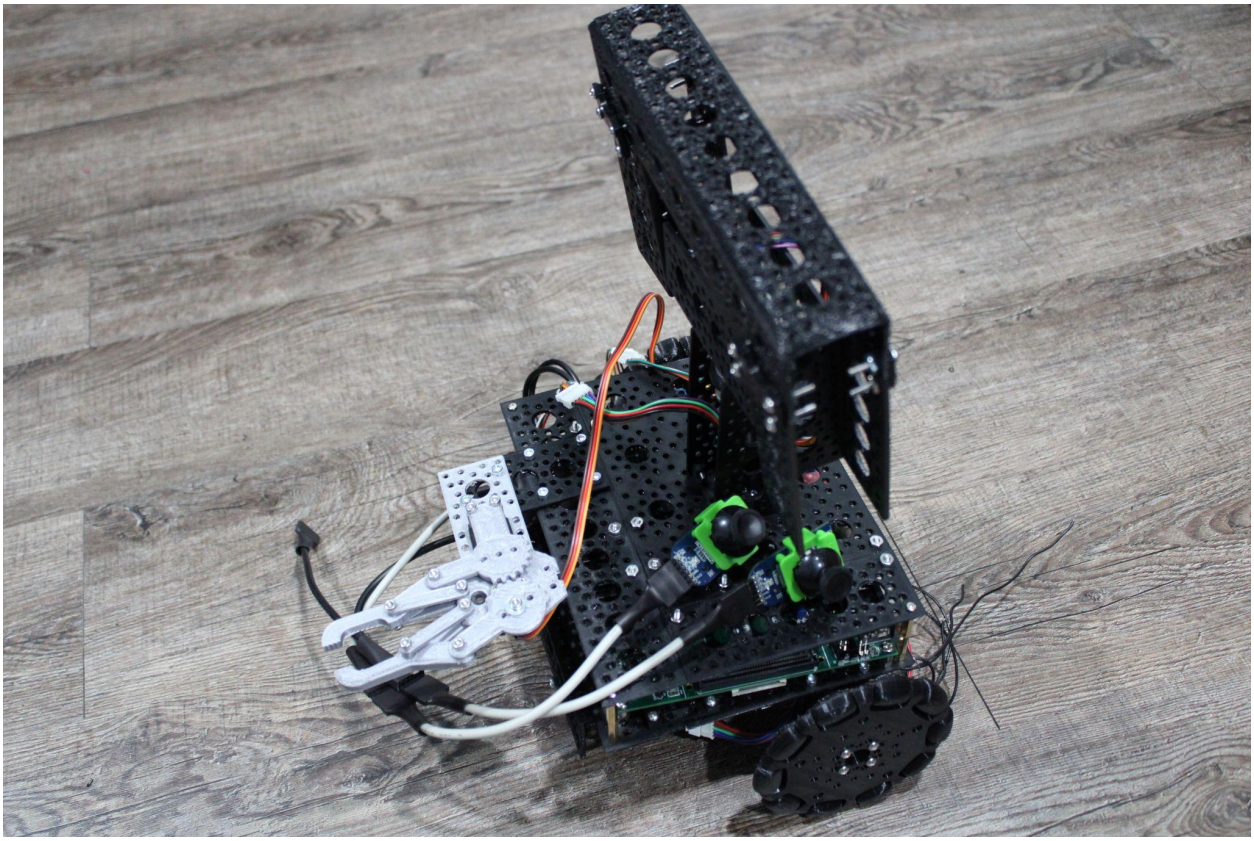

Fig. 5: Fully Assembled Car

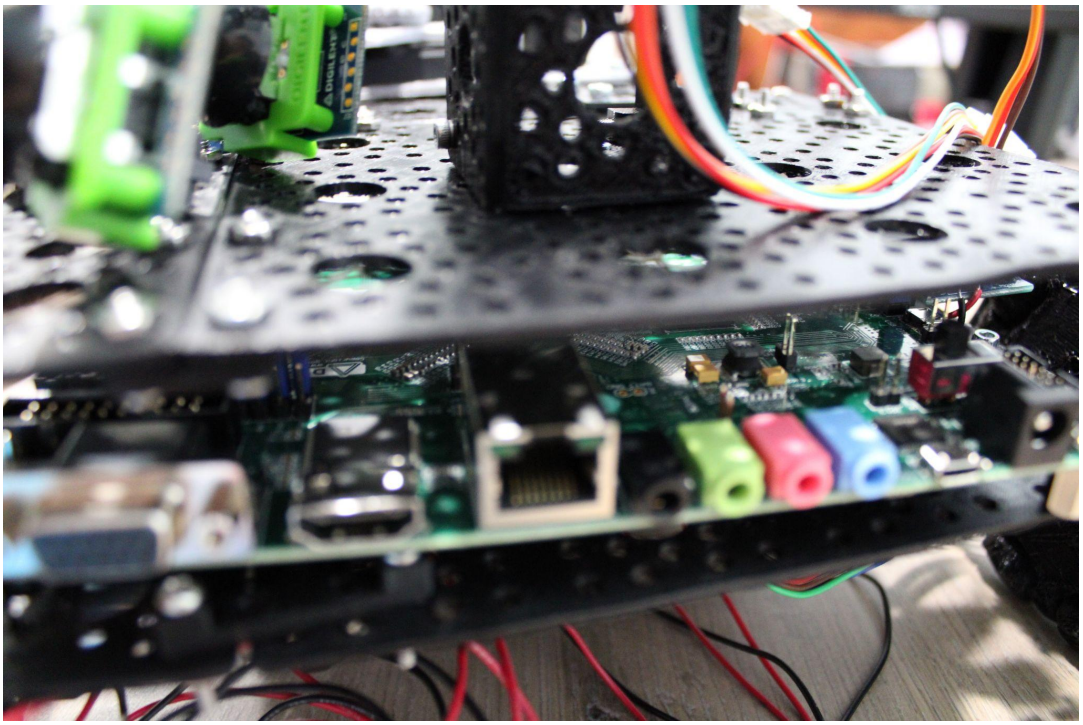

Fig. 6: Showing the Zedboard in Between Platforms

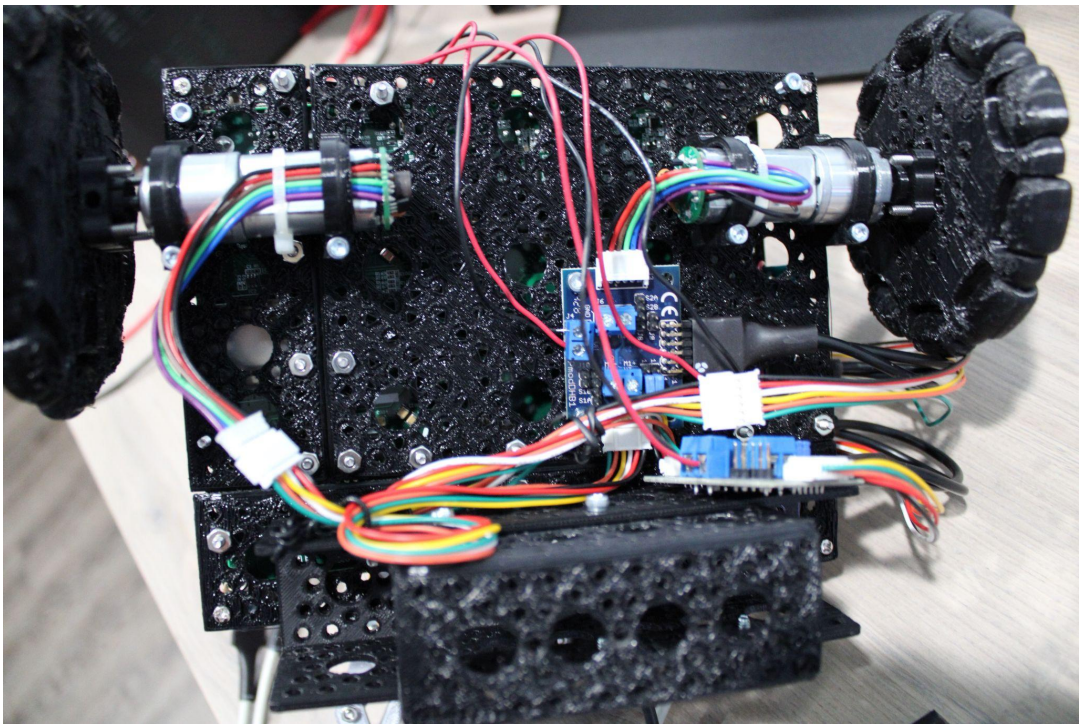

Fig. 7: Showing the Underside of the Car

To verify the PWM output, I used an oscilloscope to test if there are any signals coming out of the MIO Pmod of the Zedboard. Figure 8 shows three PWM outputs from the JE port.

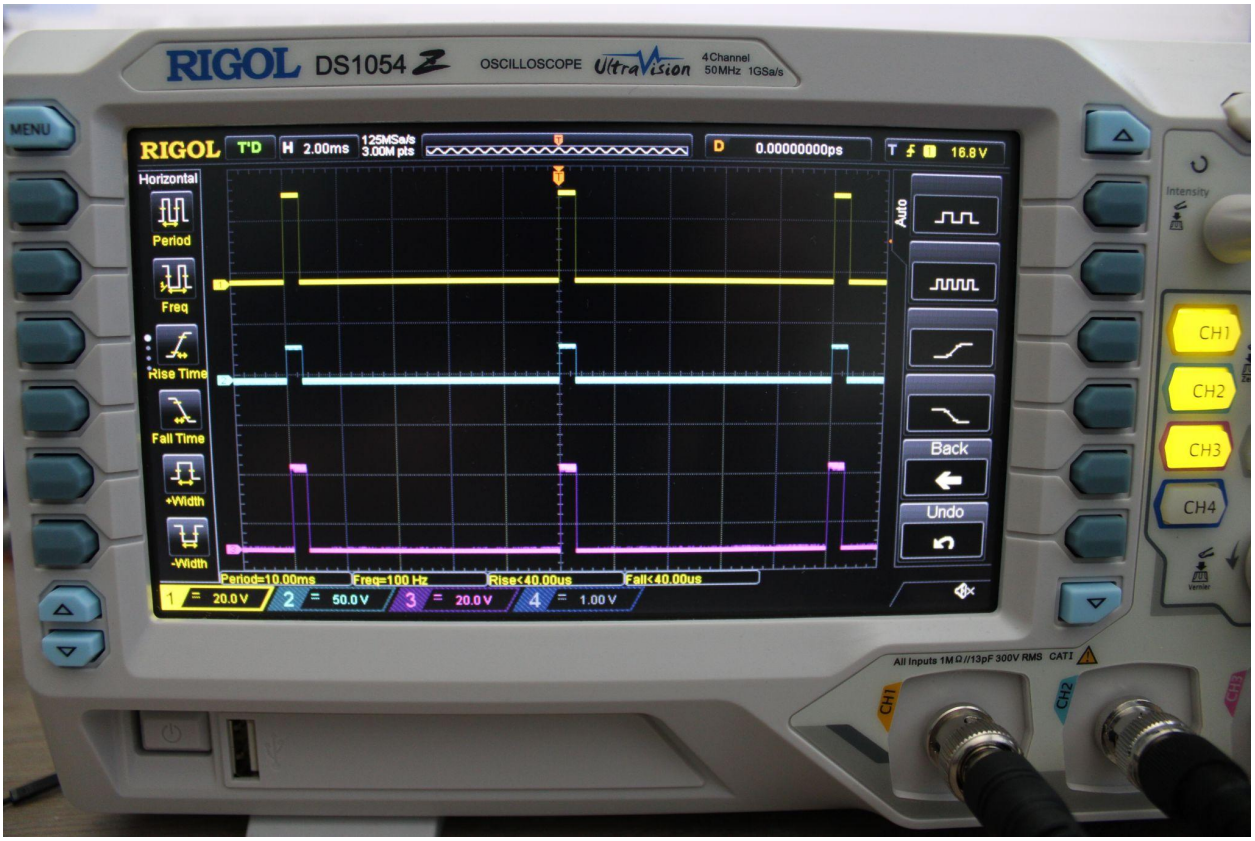

Fig. 8: Checking PWM with Oscilloscope

Video Demonstration Links:

- Zedbot Short Demonstration: https://youtu.be/0ILeQoVi7fs
- Zedbot Second Short Demonstration: https://youtu.be/ALcJU5NfLdA

References

ServoCity STEP File Library: https://www.servocity.com/step-files/
3D Model for Arm Claw Gripper: https://www.thingiverse.com/thing:969447
Interfacing Digilent Pmod IPs:
https://reference.digilentinc.com/learn/programmable-logic/tutorials/pmod-ips/start
Zedboard Reference Manual and Hardware User Guide:
https://reference.digilentinc.com/programmable-logic/zedboard/reference-manual

Appendix:

Robot Control Main Source C Code:

```
/* Aaron Joseph Nanas
*  ECE 520
*  Professor Mirzaei
*  Final Project: Robot Car with Arm
*
*  Note: Connections are as follows:
*  -- First Pmod DHB1: Horizontal and Vertical Axes Motors
*  -- Second Pmod DHB1: Wheels
*  -- First Pmod JSTK2: Controls the Horizontal and Vertical Axes Motors
*  -- Second Pmod JSTK2: Controls the wheels
*  -- Servo (Pmod MIO JE1): Controls the gripper
*/

// Include Files
#include <stdio.h>
#include <xparameters.h>
#include <xgpiops.h>
#include <xgpio.h>
#include <xscugic.h>
#include <xil_exception.h>
#include <xstatus.h>
#include <PmodJSTK2.h>
#include <MotorFeedback.h>
#include <PmodDHB1.h>
#include <PWM.h>
#include <sleep.h>
#include <xil_cache.h>
#include <xil_printf.h>

// Base addresses for button ISR
#define INTC_DEVICE_ID                          XPAR_PS7_SCUGIC_0_DEVICE_ID
#define BTNS_DEVICE_ID                          XPAR_AXI_GPIO_0_DEVICE_ID
#define INTC_GPIO_INTERRUPT_ID
XPAR_FABRIC_AXI_GPIO_0_IP2INTC_IRPT_INTR
#define BTN_INT                                 XGPIO_IR_CH1_MASK

// Base addresses for first Pmod DHB1
#define GPIO_BASEADDR_1      XPAR_PMODDHB1_0_AXI_LITE_GPIO_BASEADDR
#define PWM_BASEADDR_1       XPAR_PMODDHB1_0_PWM_AXI_BASEADDR
#define MOTOR_FB_BASEADDR_1  XPAR_PMODDHB1_0_MOTOR_FB_AXI_BASEADDR
```

```
// Base addresses for second Pmod DHB1
#define GPIO_BASEADDR_2      XPAR_PMODDHB1_1_AXI_LITE_GPIO_BASEADDR
#define PWM_BASEADDR_2       XPAR_PMODDHB1_1_PWM_AXI_BASEADDR
#define MOTOR_FB_BASEADDR_2  XPAR_PMODDHB1_1_MOTOR_FB_AXI_BASEADDR

#define CLK_FREQ 100000000 // FCLK0 frequency not found in xparameters.h
#define CPU_CLOCK_FREQ_HZ (XPAR_PS7_CORTEXA9_0_CPU_CLK_FREQ_HZ)

#define JE1_MIO 13

// Variables for first Pmod DHB1
#define PWM_PER_1                2
#define SENSOR_EDGES_PER_REV_1   4
#define GEARBOX_RATIO_1          53

// Variables for second Pmod DHB1
#define PWM_PER_2                2
#define SENSOR_EDGES_PER_REV_2   4
#define GEARBOX_RATIO_2          53

// Function Prototypes: Motor Control
void initializePMOD();
void runRobotArm();
void rotateArm();
void rotateBase();
void driveForward();
void driveBackward();
int gripperControl();
void feedbackCounter_1(int16_t sensor_edges_1);
void feedbackCounter_2(int16_t sensor_edges_2);

// Function Prototypes: Push Button ISR
int initializeButtons();
static void BTN_Intr_Handler(void *baseaddr_p);
static int InterruptSystemSetup(XScuGic *XScuGicInstancePtr);
static int IntcInitFunction(u16 DeviceId, XGpio *GpioInstancePtr);

// Global Variables and Structs Initialization
PmodJSTK2 joystick1;
PmodJSTK2 joystick2;
PmodDHB1 pmodDHB1_1;
PmodDHB1 pmodDHB1_2;
MotorFeedback motorFeedback1;
MotorFeedback motorFeedback2;
JSTK2_Position position1;
JSTK2_DataPacket rawdata1;
JSTK2_Position position2;
```

```
JSTK2_DataPacket rawdata2;
XGpioPs GpioPs;
XGpioPs_Config *ConfigPtr;
XGpio BTNInst;
XScuGic INTCInst;

static int btn_value;
static int DHB1_1_MOTOR_SPEED;
static int DHB1_2_MOTOR_SPEED;

// Calling Functions to Main
int main() {

   while(1) {
   initializePMOD();
   runRobotArm();
       }

   return 0;
}

void initializePMOD() {
   DHB1_begin(&pmodDHB1_1, GPIO_BASEADDR_1, PWM_BASEADDR_1, CLK_FREQ, PWM_PER_1);
   MotorFeedback_init(
      &motorFeedback1,
      MOTOR_FB_BASEADDR_1,
      CLK_FREQ,
      SENSOR_EDGES_PER_REV_1,
      GEARBOX_RATIO_1
   );
   DHB1_motorDisable(&pmodDHB1_1);

   DHB1_begin(&pmodDHB1_2, GPIO_BASEADDR_2, PWM_BASEADDR_2, CLK_FREQ, PWM_PER_2);
   MotorFeedback_init(
      &motorFeedback2,
      MOTOR_FB_BASEADDR_2,
      CLK_FREQ,
      SENSOR_EDGES_PER_REV_2,
      GEARBOX_RATIO_2
   );
   DHB1_motorDisable(&pmodDHB1_2);

   // Initialize the Pmod Joystick device
   JSTK2_begin(
      &joystick1,
      XPAR_PMODJSTK2_0_AXI_LITE_SPI_BASEADDR,
      XPAR_PMODJSTK2_0_AXI_LITE_GPIO_BASEADDR
```

```
   );

   // Set inversion register to invert only the Y axis
   JSTK2_setInversion(&joystick1, 0, 1);

   // Initialize Joystick 2 device
   JSTK2_begin(
      &joystick2,
      XPAR_PMODJSTK2_1_AXI_LITE_SPI_BASEADDR,
      XPAR_PMODJSTK2_1_AXI_LITE_GPIO_BASEADDR
   );

   // Set inversion register to invert only the Y axis
   JSTK2_setInversion(&joystick2, 0, 1);
}

int initializeButtons() {
       int status;

       status = XGpio_Initialize(&BTNInst, BTNS_DEVICE_ID);

       if(status != XST_SUCCESS) return XST_FAILURE;

       XGpio_SetDataDirection(&BTNInst, 1, 0xFF);

       // Initialize interrupt controller
       status = IntcInitFunction(INTC_DEVICE_ID, &BTNInst);
       if(status != XST_SUCCESS) return XST_FAILURE;

       return 0;
}

void runRobotArm() {
       DHB1_1_MOTOR_SPEED = 50;
       DHB1_2_MOTOR_SPEED = 90;
       DHB1_setMotorSpeeds(&pmodDHB1_1, DHB1_1_MOTOR_SPEED, DHB1_1_MOTOR_SPEED);
       DHB1_setMotorSpeeds(&pmodDHB1_2, DHB1_2_MOTOR_SPEED,  DHB1_2_MOTOR_SPEED);

       initializeButtons();
       MotorFeedback_clearPosCounter(&motorFeedback1);
       MotorFeedback_clearPosCounter(&motorFeedback2);
       // Get X and Y values of the Joystick
       position1 = JSTK2_getPosition(&joystick1);
       position2 = JSTK2_getPosition(&joystick2);
       // Get Joystick button data
       rawdata1 = JSTK2_getDataPacket(&joystick1);
       rawdata2 = JSTK2_getDataPacket(&joystick2);
```

```
	//xil_printf(
	//	"%s%s%s%s\r\n",
	//	(rawdata1.Jstk != 0) ? "\tJoystick 1 pressed" : "",
	//	(rawdata1.Trigger != 0) ? "\tTrigger 1 pressed" : "",
	//	(rawdata2.Jstk != 0) ? "\tJoystick 2 pressed" : "",
	//	(rawdata2.Trigger != 0) ? "\tTrigger 2 pressed" : ""
	//	);
	//usleep(50000);

	// Initialize the Joystick RGB LEDs
    if ((rawdata1.Jstk != 0 || rawdata1.Trigger != 0) && (rawdata2.Jstk == 0 ||
rawdata2.Trigger == 0)) {
	JSTK2_setLedRGB(&joystick1, 0, 255, 0);
	}
	else if ((rawdata2.Jstk != 0 || rawdata2.Trigger != 0) && (rawdata1.Jstk ==
0 || rawdata1.Trigger == 0)) {
		JSTK2_setLedRGB(&joystick2, 0, 255, 0);
	}
	else {
		JSTK2_setLedRGB(&joystick1, position1.XData, 0, position1.YData);
		JSTK2_setLedRGB(&joystick2, position2.XData, 0, position2.YData);
	}
	driveForward();
	driveBackward();
	rotateArm();
	rotateBase();
	gripperControl();
}

void rotateArm() {
	DHB1_setMotorSpeeds(&pmodDHB1_1, 0, 50);
	if ((position1.XData == 128) && (position1.YData >= 150)) { // Checks for
when the first joystick's direction is up
		DHB1_motorDisable(&pmodDHB1_1);
		usleep(6);
		DHB1_setDirs(&pmodDHB1_1, 0, 0); // Move the arm forward
		feedbackCounter_1(5);
	}

	else if ((position1.XData == 128) && (position1.YData <= 100)) { // Checks
for when the first joystick's direction is down
		DHB1_motorDisable(&pmodDHB1_1);
		usleep(6);
		DHB1_setDirs(&pmodDHB1_1, 1, 1); // Move the arm backward
		feedbackCounter_1(5);
```

```
	}
}

void rotateBase() {
	DHB1_setMotorSpeeds(&pmodDHB1_1, 50, 0);
	if ((position1.XData <= 100) && (position1.YData == 128)) { // Checks for
when the first joystick's direction is left
		DHB1_motorDisable(&pmodDHB1_1);
		usleep(6);
		DHB1_setDirs(&pmodDHB1_1, 1, 1); // Rotate the base counter-clockwise
		feedbackCounter_1(5);
	}

	else if ((position1.XData >= 150) && (position1.YData == 128)) { // Checks
for when the first joystick's direction is right
		DHB1_motorDisable(&pmodDHB1_1);
		usleep(6);
		DHB1_setDirs(&pmodDHB1_1, 0, 0); // Rotate the base clockwise
		feedbackCounter_1(5);
	}
}

void driveForward() {
	 if ((position2.XData == 128) && (position2.YData >= 129)) { // Checks for
when the second joystick's direction is up
		DHB1_motorDisable(&pmodDHB1_2); // Disable the PWM before the
motors change direction
		usleep(6);                    // Adding a short delay to prevent
short circuit
		DHB1_setDirs(&pmodDHB1_2, 1, 1); // Set direction forward
		feedbackCounter_2(240); // Motor runs until the specified amount of
positive edges is reached
	 }
}

void driveBackward() {
	 if ((position2.XData == 128) && (position2.YData <= 127)) { // Checks for
when the second joystick's direction is down
		DHB1_motorDisable(&pmodDHB1_2); // Disable the PWM before the
motors change direction
		usleep(6);                    // Adding a short delay to prevent
short circuit
		DHB1_setDirs(&pmodDHB1_2, 0, 0); // Set direction backward
		feedbackCounter_2(240); // Motor runs until the specified amount of
positive edges is reached
	  }
}
```

```
// Function to drive the servo using the Pmod MIO JE1 port
// Uses usleep function to generate the frequency and duty cycle
int gripperControl() {
	int Status;

	ConfigPtr = XGpioPs_LookupConfig(XPAR_XGPIOPS_0_DEVICE_ID);
	Status = XGpioPs_CfgInitialize(&GpioPs, ConfigPtr,
	ConfigPtr->BaseAddr);

	if (Status != XST_SUCCESS) {
		return XST_FAILURE;
	}

	// Set directions and output enable for PMOD JE1 Pin
	XGpioPs_SetDirectionPin(&GpioPs, JE1_MIO, 1);
	XGpioPs_SetOutputEnablePin(&GpioPs, JE1_MIO, 1);

	if (rawdata1.Trigger == 1) {
		XGpioPs_WritePin(&GpioPs, JE1_MIO,1);
		usleep(1000); // Set output high for 1 ms (0 degrees)
		XGpioPs_WritePin(&GpioPs, JE1_MIO,0);
		usleep(20000); // Set output low for the rest of the period
	}

	else if (rawdata2.Trigger == 1) {
		XGpioPs_WritePin(&GpioPs, JE1_MIO, 1);
		usleep(2000); // Set output high for 2 ms (180 degrees)
		XGpioPs_WritePin(&GpioPs, JE1_MIO, 0);
		usleep(20000); // Set output low for the rest of the period
	}

	else {
		XGpioPs_WritePin(&GpioPs, JE1_MIO, 0);
		usleep(1000);
	}

	return XST_SUCCESS;
}

void feedbackCounter_1(int16_t sensor_edges_1) {
   DHB1_motorEnable(&pmodDHB1_1);
   int16_t dist_1 = MotorFeedback_getDistanceTraveled(&motorFeedback1);
   while (dist_1 < sensor_edges_1) {
      dist_1 = MotorFeedback_getDistanceTraveled(&motorFeedback1);
   }
   MotorFeedback_clearPosCounter(&motorFeedback1);
```

```
   DHB1_motorDisable(&pmodDHB1_1);
}

void feedbackCounter_2(int16_t sensor_edges_2) {
   DHB1_motorEnable(&pmodDHB1_2);
   int16_t dist_2 = MotorFeedback_getDistanceTraveled(&motorFeedback2);
   while (dist_2 < sensor_edges_2) {
      dist_2 = MotorFeedback_getDistanceTraveled(&motorFeedback2);
   }
   MotorFeedback_clearPosCounter(&motorFeedback2);
   DHB1_motorDisable(&pmodDHB1_2);
}

void BTN_Intr_Handler(void *InstancePtr)
{
       // Disable GPIO interrupts
       XGpio_InterruptDisable(&BTNInst, BTN_INT);
       // Ignore additional button presses
       if ((XGpio_InterruptGetStatus(&BTNInst) & BTN_INT) !=
                    BTN_INT) {
                    return;
             }
       btn_value = XGpio_DiscreteRead(&BTNInst, 1);

       if (btn_value == 1) {
             xil_printf("\nCurrent Motor Speeds: \n"
                          "DHB1_1 Motors: %d\n"
                          "DHB1_2 Motors: %d\n",DHB1_1_MOTOR_SPEED,
DHB1_2_MOTOR_SPEED );
       }

       else if (btn_value == 4) {
             xil_printf("\nLeft button (BTNL) was pressed. Decreasing DHB1_1 Motor
Speed.\n");
             DHB1_1_MOTOR_SPEED = DHB1_1_MOTOR_SPEED - 5;
             xil_printf("\nCurrent Speed of DHB1_1 Motors (Arm/Base): %d\n",
DHB1_1_MOTOR_SPEED);
       }

       else if (btn_value == 8) {
             xil_printf("\nRight button (BTNR) was pressed. Increasing DHB1_1 Motor
Speed.\n");
             DHB1_1_MOTOR_SPEED = DHB1_1_MOTOR_SPEED + 5;
             xil_printf("\nCurrent Speed of DHB1_1 Motors (Arm/Base): %d\n",
DHB1_1_MOTOR_SPEED);
       }
```

```
	else if (btn_value == 16) {
		xil_printf("\nUp button (BTNU) was pressed. Increasing DHB1_2 Motor
Speed.\n");
		DHB1_2_MOTOR_SPEED = DHB1_2_MOTOR_SPEED + 5;
		xil_printf("\nCurrent Speed of DHB1_2 Motors (Wheels): %d\n",
DHB1_2_MOTOR_SPEED);
	}

	else if (btn_value == 2) {
		xil_printf("\nDown button (BTND) was pressed. Decreasing DHB1_2 Motor
Speed.\n");
		DHB1_2_MOTOR_SPEED = DHB1_2_MOTOR_SPEED - 5;
		xil_printf("\nCurrent Speed of DHB1_2 Motors (Wheels): %d\n",
DHB1_2_MOTOR_SPEED);
	}

    (void)XGpio_InterruptClear(&BTNInst, BTN_INT);
    // Enable GPIO interrupts
    XGpio_InterruptEnable(&BTNInst, BTN_INT);
}

int InterruptSystemSetup(XScuGic *XScuGicInstancePtr)
{
	// Enable interrupt
	XGpio_InterruptEnable(&BTNInst, BTN_INT);
	XGpio_InterruptGlobalEnable(&BTNInst);
	Xil_ExceptionRegisterHandler(XIL_EXCEPTION_ID_INT,

(Xil_ExceptionHandler)XScuGic_InterruptHandler,
                                                    XScuGicInstancePtr);
	Xil_ExceptionEnable();
	return XST_SUCCESS;
}
int IntcInitFunction(u16 DeviceId, XGpio *GpioInstancePtr)
{
	XScuGic_Config *IntcConfig;
	int status;

	// Interrupt controller initialization
	IntcConfig = XScuGic_LookupConfig(DeviceId);
	status = XScuGic_CfgInitialize(&INTCInst, IntcConfig,
IntcConfig->CpuBaseAddress);

	if(status != XST_SUCCESS)
		return XST_FAILURE;

	// Call to interrupt setup
```

```
	status = InterruptSystemSetup(&INTCInst);
	if(status != XST_SUCCESS)
		return XST_FAILURE;

	// Connect GPIO interrupt to handler
	status = XScuGic_Connect(&INTCInst,
	 INTC_GPIO_INTERRUPT_ID,
	 (Xil_ExceptionHandler)BTN_Intr_Handler,
	 (void *)GpioInstancePtr);

	if(status != XST_SUCCESS)
		return XST_FAILURE;

	// Enable GPIO interrupts interrupt
	XGpio_InterruptEnable(GpioInstancePtr, 1);
	XGpio_InterruptGlobalEnable(GpioInstancePtr);

	// Enable GPIO and timer interrupts in the controller
	XScuGic_Enable(&INTCInst, INTC_GPIO_INTERRUPT_ID);

	return XST_SUCCESS;
}
```