5-1-2020

DRIVING THE IROBOT CREATE 2: COMMAND FUNCTIONS

Sophie Soueid

DRIVING THE IROBOT CREATE 2:

COMMAND FUNCTIONS

by

SOPHIE ELIZABETH SOUEID

Presented to the Faculty of the Honors College of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

HONORS BACHELOR OF SCIENCE IN ELECTRICAL ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

May 2020

ACKNOWLEDGMENTS

First and foremost, I would like to acknowledge Zachary Holloway, my colleague and friend, who developed the other half of this driver. Although the immense amount of time we have spent together on various projects has caused us to butt heads at times, I could not have found another colleague who strives for excellence in the same way that he does.

I want to thank my faculty mentor, Dr. Greg Turner, not just for all the academic opportunities he provided me the past four years, but also for being a source of inspiration in the form of character. I will forever strive to be as kind, dedicated, and patient as he was to me. I also want to thank Dr. Howard Russell, who has always held great trust in my knowledge and ability and has helped me develop my passion for teaching others.

I would also like to acknowledge my fellow friends and associates in Tau Beta Pi for constantly challenging me and for acting almost as a second family. I would especially like to thank Maxwell Sanders for being the first student leader I truly looked up to. I hope to one day be as authentic and considerate of a leader as he always was to me.

Lastly, I would like to thank my father, who I, without a doubt, took after regarding our STEM-oriented minds. Dad, you were always my first inspiration in engineering. When you passed, my goals in life solidified. I wanted to do things that would make you proud, and the first obvious goal became getting an engineering degree. Eleven years later, it still seems bittersweet to finally achieve this goal without you. Nonetheless, I will persevere, and I know the next step for me will be finding another goal that will make you proud.

May 4, 2020

ABSTRACT

DRIVING THE IROBOT CREATE 2:

COMMAND FUNCTIONS

Sophie Soueid, B.S. Electrical Engineering

The University of Texas at Arlington, 2020

Faculty Mentor: Greg Turner

A driver library in the C programming language was developed to interface between the iRobot Create® 2 Programmable Robot's command functions and the MSP432P401R via Universal Asynchronous Receiver/Transmitter (UART). This allows for quick and simple development of microcontroller control over the Create 2's command functions. It utilizes information from the iRobot Open Interface to reduce the complicated series of data in the form of arrays that have to be sent to the iRobot microcontroller down to individual functions. The driver that interfaces between these two devices greatly increases the ease of access to hobbyist programmers using the Create 2 and MSP432P401R. This driver is written in the C programming language, and so it can be used in any project that uses either C or C++ as its coding language for the primary functions. The project has been published on Hackster.io for public use.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Motivation

The iRobot Create® 2 is a Roomba® product marketed at hobbyist programmers interested in the iRobot line of products. The iRobot Open Interface enables these amateur programmers to interface their iRobot Create 2 with any microcontroller capable of communicating over Universal Asynchronous Receive/Transmit (UART), which is an extremely basic form of communication available on almost any microcontroller on the market. One of these microcontrollers is the MSP432P401R, a microcontroller made by Texas Instruments that is easy to use for the amateur programmer and has a wide database of documentation and examples that are easily accessible. However, many of the functions of the iRobot Create 2 available to the end user require background knowledge of binary strings and bitmasks, which are usually only taught while taking a degree in electrical or computer engineering. By creating a driver to interface between the MSP432P401R and the iRobot Create 2, the hobbyist programmers who make up the iRobot Create 2's target audience will easily be able to access and use more complex functions of the device to use the product to its fullest potential.

1.2 Senior Design and the Creation of the Driver

A senior design group was assigned a senior design project in the Spring 2019 semester called the Pet Waste Avoiding Autonomous Vacuum (PWAAV) under our mentor Dr. Greg Turner. This project required us to use an iRobot Create 2 in conjunction

with an MSP432P401R microcontroller and the Pixy2 color recognition camera, in addition to other optional sensors, in order to create an autonomous vacuum capable of detecting and avoiding pet waste around the house and give the homeowner a wireless notification that the vacuum has encountered pet waste. This driver enabled easier development of code for this senior design project in addition to its primary purpose of providing a more accessible way to program the iRobot Create 2 for amateur programmers. It enabled both the senior design group to achieve their project's goal in addition to providing an ease of access to those just beginning to explore the world of programmable microcontrollers.

CHAPTER 2

DRIVER DESIGN AND DEVELOPMENT

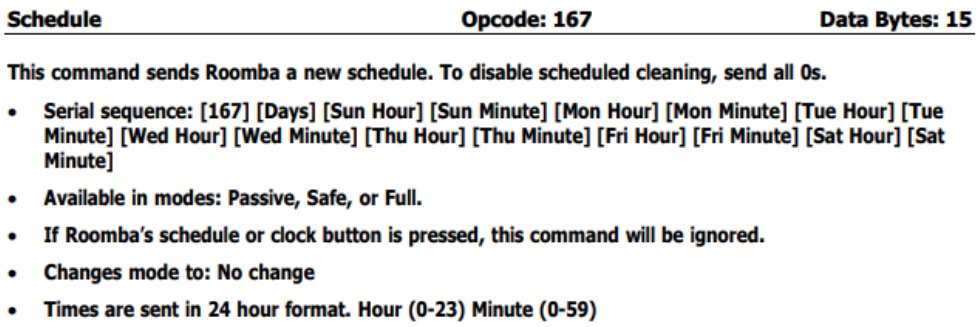
2.1 The sendUART Function

The basis of the entire driver is a function called sendUART, which enables the MSP432P401R to send information to the Create® 2. This function sends data using Universal Asynchronous Transmit/Receive (UART), which is a communication protocol where two devices agree on a predetermined speed to communicate data as opposed to using a shared clock line to time data transmissions. The two devices communicate at a baud rate of 115200, which is the default speed of communication for the Create 2. All the command functions in the driver utilize this sendUART function in order to initiate communication and let the Create 2 know what command is to be performed. This function will always begin with the opcode of the command trying to be issued, followed by any other data that may be necessary for the Create 2 to know the details of how to perform that command.

2.2 Formatting Data for Commands

Before any data is sent to the Create 2 through the sendUART function, the supporting data for each command must be passed to the function associated with a command and formatted in a way that the microcontroller onboard the Create 2 can parse. These values are put into an array that will be sent by the sendUART function. In order to make the driver more useful to the Create 2's target demographic of amateur programmers,

the functions had to be written so these programmers could easily utilize them without needing a strong background in electrical or computer engineering.



Schedule **Opcode: 167** **Data Bytes: 15**

This command sends Roomba a new schedule. To disable scheduled cleaning, send all 0s.

- **Serial sequence: [167] [Days] [Sun Hour] [Sun Minute] [Mon Hour] [Mon Minute] [Tue Hour] [Tue Minute] [Wed Hour] [Wed Minute] [Thu Hour] [Thu Minute] [Fri Hour] [Fri Minute] [Sat Hour] [Sat Minute]**
- **Available in modes: Passive, Safe, or Full.**
- **If Roomba's schedule or clock button is pressed, this command will be ignored.**
- **Changes mode to: No change**
- **Times are sent in 24 hour format. Hour (0-23) Minute (0-59)**

Figure 2.1: Image from the Create 2 Open Interface

For example, the schedule command requires a complicated payload to be attached with the days being represented by the value of a binary string. Because many amateur programmers may not have a strong grasp on formatting binary strings that correspond to less intuitive things like days of the week, the command was made simpler for amateur programmers by limiting the changes in schedule to only affect one day per function call.

Table 2.1: Table of Binary String for Schedule

Days

Bit	7	6	5	4	3	2	1	0
Value	Reserved	Sat	Fri	Thu	Wed	Tue	Mon	Sun

In addition, the days of the week are no longer represented by binary numbers, but instead by a variable type called enums that allow users to input a word (for example, MONDAY) and have that name correspond to the binary value that the Create 2 will accept.

All functions were written in a similar manner with the end user in mind. This also meant that code was developed such that the inexperienced end user could not send commands to the Create 2 that would break it or that would cause memory failures in the microcontroller.

2.3 Sending Data to the Create 2

All the necessary data for executing a command function is loaded into an array containing 8-bit values named the command buffer. However, UART is only capable of sending a single 8-bit value in a transmission. In order to send this over UART, the command buffer is loaded into a first-in first-out (FIFO) buffer, which sends the individual 8-bit values in order when the previous transmission has finished. After receiving the opcode and all the necessary supporting data, the Create 2 will carry out the requested command. Because these are only command functions, it is largely not necessary to prepare a receiving buffer, as no data is sent back in the command functions except for a few functions that simply command the Create 2 to send a stream of packets from the data functions.

CHAPTER 3

CONCLUSION

The command functions provided through the driver make the Create 2 much easier to use in conjunction with the MSP432P401R, especially for amateur and hobbyist programmers. The complex code normally required to be developed by the end user for command functions is all provided in this portion of the driver, and the end user has a much more intuitive way to drive the Create 2. This driver also significantly helped the PWAAV senior design team develop code that allowed us to drive and demonstrate our senior design project. While this code is currently written for the MSP432P401R in C and C++ specifically, it could very easily be adapted to work with any other C-based microcontroller with very little effort, only requiring changes to parts of the code specific to the microcontroller, such as the UART commands or FIFO buffer.

APPENDIX A

HEADER CODE LISTING

```
/*
 * Create2_UART_Header
 *
 *   Created on: August 21, 2019
 *      Authors: Zachary Holloway and Sophie Soueid
 */

#ifndef CREATE2_UART_HEADER_H_
#define CREATE2_UART_HEADER_H_

//*****************************************************************************
//
// If building with a C++ compiler, make all of the definitions in this header
// have a C binding.
//
//*****************************************************************************
#ifdef __cplusplus
extern "C"
{
#endif

#include <stdint.h>

#define CREATE2_OPCODE_START                    (128)
#define CREATE2_OPCODE_RESET                    (7)
#define CREATE2_OPCODE_STOP                     (173)
#define CREATE2_OPCODE_BAUD                     (129)
#define CREATE2_OPCODE_SAFE                     (131)
#define CREATE2_OPCODE_FULL                     (132)
#define CREATE2_OPCODE_CLEAN                    (135)
#define CREATE2_OPCODE_MAX                      (136)
#define CREATE2_OPCODE_SPOT                     (134)
#define CREATE2_OPCODE_SEEK_DOCK                (143)
#define CREATE2_OPCODE_POWER                    (133)
#define CREATE2_OPCODE_SCHEDULE                 (167)
#define CREATE2_OPCODE_SET_DAY_TIME             (168)
#define CREATE2_OPCODE_DRIVE                    (137)
#define CREATE2_OPCODE_DRIVE_DIRECT             (145)
#define CREATE2_OPCODE_DRIVE_PWM                (146)
#define CREATE2_OPCODE_MOTORS                   (138)
#define CREATE2_OPCODE_PWM_MOTORS               (144)
#define CREATE2_OPCODE_LEDS                     (139)
#define CREATE2_OPCODE_SCHEDULING_LEDS          (162)
#define CREATE2_OPCODE_DIGIT_LEDS_RAW           (163)
#define CREATE2_OPCODE_BUTTONS                  (165)
#define CREATE2_OPCODE_DIGIT_LEDS_ASCII         (129)
#define CREATE2_OPCODE_SONG                     (140)
#define CREATE2_OPCODE_PLAY                     (141)
#define CREATE2_OPCODE_SENSORS                  (142)
#define CREATE2_OPCODE_QUERY_LIST               (149)
#define CREATE2_OPCODE_STREAM                   (148)
#define CREATE2_OPCODE_PAUSE_RESUME_STREAM      (150)
#define CREATE2_PACKET_BUMP_WHEEL_DROPS         (7)
#define CREATE2_PACKET_WALL                     (8)
#define CREATE2_PACKET_CLIFF_LEFT               (9)
#define CREATE2_PACKET_CLIFF_FRONT_LEFT         (10)
```

```
#define CREATE2_PACKET_CLIFF_FRONT_RIGHT            (11)
#define CREATE2_PACKET_CLIFF_RIGHT                  (12)
#define CREATE2_PACKET_VIRTUAL_WALL                 (13)
#define CREATE2_PACKET_WHEEL_OVERCURRENTS           (14)
#define CREATE2_PACKET_DIRT_DETECT                  (15)
#define CREATE2_PACKET_UNUSED_BYTE                  (16)
#define CREATE2_PACKET_INFRARED_CHARACTER_OMNI      (17)
#define CREATE2_PACKET_BUTTONS                      (18)
#define CREATE2_PACKET_DISTANCE                     (19)
#define CREATE2_PACKET_ANGLE                        (20)
#define CREATE2_PACKET_CHARGING_STATE               (21)
#define CREATE2_PACKET_VOLTAGE                      (22)
#define CREATE2_PACKET_CURRENT                      (23)
#define CREATE2_PACKET_TEMPERATURE                  (24)
#define CREATE2_PACKET_BATTERY_CHARGE               (25)
#define CREATE2_PACKET_BATTERY_CAPACITY             (26)
#define CREATE2_PACKET_WALL_SIGNAL                  (27)
#define CREATE2_PACKET_CLIFF_LEFT_SIGNAL            (28)
#define CREATE2_PACKET_CLIFF_FRONT_LEFT_SIGNAL      (29)
#define CREATE2_PACKET_CLIFF_FRONT_RIGHT_SIGNAL     (30)
#define CREATE2_PACKET_CLIFF_RIGHT_SIGNAL           (31)
#define CREATE2_PACKET_CHARGING_SOURCES_AVAILABLE   (34)
#define CREATE2_PACKET_OI_MODE                      (35)
#define CREATE2_PACKET_SONG_NUMBER                  (36)
#define CREATE2_PACKET_SONG_PLAYING                 (37)
#define CREATE2_PACKET_NUMBER_STREAM_PACKETS        (38)
#define CREATE2_PACKET_REQUESTED_VELOCITY           (39)
#define CREATE2_PACKET_REQUESTED_RADIUS             (40)
#define CREATE2_PACKET_REQUESTED_RIGHT_RADIUS       (41)
#define CREATE2_PACKET_REQUESTED_LEFT_RADIUS        (42)
#define CREATE2_PACKET_LEFT_ENCODER_COUNTS          (43)
#define CREATE2_PACKET_RIGHT_ENCODER_COUNTS         (44)
#define CREATE2_PACKET_LIGHT_BUMPER                 (45)
#define CREATE2_PACKET_LIGHT_BUMP_LEFT_SIGNAL       (46)
#define CREATE2_PACKET_LIGHT_BUMP_FRONT_LEFT_SIGNAL (47)
#define CREATE2_PACKET_LIGHT_BUMP_CENTER_LEFT_SIGNAL (48)
#define CREATE2_PACKET_LIGHT_BUMP_CENTER_RIGHT_SIGNAL (49)
#define CREATE2_PACKET_LIGHT_BUMP_FRONT_RIGHT_SIGNAL (50)
#define CREATE2_PACKET_LIGHT_BUMP_RIGHT_SIGNAL      (51)
#define CREATE2_PACKET_LEFT_MOTOR_CURRENT           (54)
#define CREATE2_PACKET_RIGHT_MOTOR_CURRENT          (55)
#define CREATE2_PACKET_MAIN_BRUSH_MOTOR_CURRENT     (56)
#define CREATE2_PACKET_SIDE_BRUSH_MOTOR_CURRENT     (57)
#define CREATE2_PACKET_STASIS                       (58)

#define CREATE2_PACKET_INFRARED_CHARACTER_LEFT      (52)
#define CREATE2_PACKET_INFRARED_CHARACTER_RIGHT     (53)

typedef enum day {
   SUNDAY = 1,
   MONDAY = 2,
   TUESDAY = 4,
   WEDNESDAY = 8,
   THURSDAY = 16,
   FRIDAY = 32,
   SATURDAY = 64
```

```
} DAY_VALUE;

extern DAY_VALUE;


typedef enum mainBrush {
   OFF = 0,
   INWARD = 1,
   OUTWARD =2
} MOTORS_MAIN_BRUSH;

extern MOTORS_MAIN_BRUSH;


typedef enum sideBrush {
   OFF = 0,
   CCW = 1,
   CW = 2
} MOTORS_SIDE_BRUSH;

extern MOTORS_SIDE_BRUSH;


typedef enum vacuum {
   OFF = 0,
   ON = 1
} MOTORS_VACUUM;

extern MOTORS_VACUUM;


typedef enum digit {
   FAR_LEFT = 1,
   MIDDLE_LEFT = 2,
   MIDDLE_RIGHT = 3,
   FAR_RIGHT = 4
} DIGIT_POSITION;

extern DIGIT_POSITION;


typedef enum {
  CLEAN = 1,
  SPOT = 2,
  DOCK = 4,
  MINUTE = 8,
  HOUR = 16,
  DAY = 32,
  SCHEDULE = 64,
  CLOCK = 128,
  ALL = 255
} BUTTON;

extern BUTTON button;

typedef enum {
```

```
  LIGHT_BUMPER_LEFT = 1,
  LIGHT_BUMPER_FRONT_LEFT = 2,
  LIGHT_BUMPER_CENTER_LEFT = 4,
  LIGHT_BUMPER_CENTER_RIGHT = 8,
  LIGHT_BUMPER_FRONT_RIGHT = 16,
  LIGHT_BUMPER_RIGHT = 32,
  ALL_BUMPER = 63
} LIGHTBUMPER;

extern LIGHTBUMPER lightBumper;

//*****************************************************************************
extern void configUART(void);
//*****************************************************************************
//
// @brief: This function configures all the necessary settings for UART A2.
//
// @return: None.
//
//*****************************************************************************
extern void sendUART(void);
//*****************************************************************************
//
// @brief: This function is used to send all data packets to the Create 2 by assigning
//       values to an integer array and filling the TX buffer with data to be sent.
//
// @return: None.
//
//*****************************************************************************
extern void roombaStart(void);
//*****************************************************************************
//
// @brief:  This command starts the open interface (OI). You must always send the
//        Start command before sending any other commands to the OI. This command
//        also sets the mode to passive.
//
// @return: None.
//
//*****************************************************************************
extern void roombaReset(void);
//*****************************************************************************
//
// @brief:  This command resets the robot, as if you had removed and reinserted the
//        battery.
//
// @return: None.
//
//*****************************************************************************
extern void roombaStop(void);
//*****************************************************************************
//
// @brief:  This command stops the OI. All streams will stop and the robot will no
//        longer respond to commands. Use this command when you are finished working
//        with the robot.
//
// @return: None.
```

```
//
//*****************************************************************************
extern void setRoombaBaud(uint32_t valueBaud);
//*****************************************************************************
//
// @brief:  This command sets the baud rate at which OI commands and data are sent.
//
// @param: valueBaud is the control of the baud rate.
//     Valid values are 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800,
//     38400, 57600, or 115200.
//
// @return: None.
//
//*****************************************************************************
extern void roombaSafe(void);
//*****************************************************************************
//
// @brief:  This command puts the OI into Safe mode, enabling user control of Roomba.
//         It turns off all LEDs. The OI can be in Passive, Safe, or Full mode to
//         accept this command. If a safety condition occurs, Roomba reverts
//         automatically to Passive mode. Safety conditions are:
//         1. Detection of a cliff while moving forward (or moving backward with a
//           small turning radius, less than one robot radius)
//         2. Detection of a wheel drop (on any wheel).
//         3. Charger plugged in and powered.
//
// @param:  None.
//
// @return: None.
//
//*****************************************************************************
extern void roombaFull(void);
//*****************************************************************************
//
// @brief:  This command gives you complete control over Roomba by putting the OI into
//         Full mode, and turning off the cliff, wheel-drop and internal charger
//         safety features. That is, in Full mode, Roomba executes any command that
//         you send it, even if the internal charger is plugged in, or command
//         triggers a cliff or wheel drop condition.
//
// @param:  None.
//
// @return: None.
//
//*****************************************************************************
extern void roombaClean(void);
//*****************************************************************************
//
// @brief:  This command starts the default cleaning mode. This is the same as
//         pressing Roomba's Clean button, and will pause a cleaning cycle if one
//         is already in progress.
//
// @param:  None.
//
// @return: None.
//
```

```
//*****************************************************************************
extern void roombaMax(void);
//*****************************************************************************
//
// @brief:  This command starts the Max cleaning mode, which will clean until the
//          battery is dead. This command will pause a cleaning cycle if one is
//          already in progress.
//
// @param:  None.
//
// @return: None.
//
//*****************************************************************************
extern void roombaSpot(void);
//*****************************************************************************
//
// @brief:  This command starts the Spot cleaning mode. This is the same as pressing
//          Roomba's Spot button, and will pause a cleaning cycle if one is already
//          in progress.
//
// @param:  None.
//
// @return: None.
//
//*****************************************************************************
extern void roombaSeekDock(void);
//*****************************************************************************
//
// @brief:  This command directs Roomba to drive onto the dock the next time it
//          encounters the docking beams. This is the same as pressing Roomba's Dock
//          button, and will pause a cleaning cycle if one is already in progress.
//
// @param:  None.
//
// @return: None.
//
//*****************************************************************************
extern void roombaPower(void);
//*****************************************************************************
//
// @brief:  This command powers down Roomba. The OI can be in Passive, Safe, or Full
//          mode to accept this command.
//
// @param:  None.
//
// @return: None.
//
//*****************************************************************************
extern void roombaSchedule (DAY_VALUE day, bool enable, uint8_t hour, uint8_t minute)
//*****************************************************************************
//
// @brief:  This command sends Roomba a new schedule for cleaning on a chosen day. If
//          Roomba's schedule or clock button is pressed, this command will be ignored.
//
// @param:  day will indicate which weekday's schedule is being changed.
//      Valid values are SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, and
```

```
//     SATURDAY.
//
// @param:  enable will control whether the chosen day's scheduled cleaning will occur.
//     Valid values are false (to disable the chosen day's cleaning) or true (to enable
//     the chosen day's cleaning).
//
// @param:  hour will set the hour at which the chosen day's scheduled cleaning will occur.
//     Valid values are between 0-23 inclusive. Note that hours must be represented
//     in the 24-hour format. If you wish to disable the chosen day's cleaning, send a 0.
//
// @param:  minute will set the minute at which the chosen day's scheduled cleaning will
//         occur.
//     Valid values are between 0-59 inclusive. If you wish to disable the chosen day's
//     cleaning, send a 0.
//
// @return: None.
//
//*****************************************************************************
extern void roombaScheduleDisableAll(void);
//*****************************************************************************
//
// @brief:  This command clears and disables all of the Roomba's cleaning schedules.
//
// @param:  None.
//
// @return: None.
//
//*****************************************************************************
extern void roombaSetDayTime(DAY_VALUE day, uint8_t hour, uint8_t minute);
//*****************************************************************************
//
// @brief:  This command sets the weekday and time on the Roomba. Note: If Roomba's
//         schedule or clock button is pressed, this command will be ignored.
//
// @param:  day will indicate which weekday's schedule is being changed.
//     Valid values are Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, and
//     Saturday.
//
// @param:  hour will set the hour value on the Roomba's clock.
//     Valid values are between 0-23 inclusive. Note that hours must be represented
//     in the 24-hour format.
//
// @param:  minute will set the minute value on the Roomba's clock.
//     Valid values are between 0-59 inclusive.
//
// @return: None.
//
//*****************************************************************************
extern void roombaDrive (int16_t velocity, int16_t radius);
//*****************************************************************************
//
// @brief:  This command controls Roomba's drive wheels.
//
// @param:  velocity will set the average velocity of the drive wheels in millimeters
//         per second (mm/s). Positive values will make the Roomba drive forward.
//         Negative values will make the Roomba drive backward.
```

```
//      Valid values are between -500 to 500 inclusive.
//
// @param:  radius will set the radius in millimeters at which Roomba will turn. A
//          longer radius makes the Roomba drive straighter, while the shorter radius
//          makes the Roomba turn more. The radius is measured from the center of the
//          turning circle to the center of the Roomba. Positive values will make the
//          Roomba turn counter-clockwise. Negative values will make the Roomba turn
//          clockwise.
//      Valid values are between -2000 to 2000 inclusive to specify the turn radius.
//      Special case values are -32768 or 32767 to make the Roomba drive straight,
//      -1 to make the Roomba turn in place clockwise, and 1 to make the Roomba
//      turn in place counter-clockwise.
//
// @return: None.
//
//*****************************************************************************
extern void roombaDriveDirect (int16_t rightVelocity, int16_t leftVelocity);
//*****************************************************************************
//
// @brief:  This command lets you control the forward and backward motion of Roomba's
//          drive wheels independently in terms of velocity in millimeters per second
//          (mm/s). A positive value makes the wheel drive forward, while a negative
//          value makes the wheel drive backward.
//
// @param:  rightVelocity controls the velocity of the right wheel.
//      Valid values are -500 to 500 inclusive.
//
// @param:  leftVelocity controls the velocity of the left wheel.
//      Valid values are -500 to 500 inclusive.
//
// @return: None.
//
//*****************************************************************************
extern void roombaDrivePWM (int16_t rightDutyCycle, int16_t leftDutyCycle);
//*****************************************************************************
//
// @brief:  This command lets you control the forward and backward motion of Roomba's
//          drive wheels independently in terms of duty cycle. A positive duty cycle
//          makes that wheel drive forward, while a negative duty cycle makes it drive
//          backward.
//
// @param:  rightDutyCycle controls the duty cycle (%) of the right wheel.
//      Valid values are -100 to 100 inclusive.
//
// @param:  leftDutyCycle controls the duty cycle (%) of the left wheel.
//      Valid values are -100 to 100 inclusive.
//
// @return: None.
//
//*****************************************************************************
extern void motors (MOTORS_MAIN_BRUSH mainBrush, MOTORS_SIDE_BRUSH sideBrush,
MOTORS_VACUUM vacuum);
//*****************************************************************************
//
// @brief:  This command lets you control the forward and backward motion of Roomba's
//          main brush, side brush, and vacuum independently. Motor velocity cannot
```

```
//         be controlled with this command, all motors will run at maximum speed when
//         enabled.
//
// @param:  mainBrush controls the Roomba's main brush.
//      Valid values are:
//         OFF to turn the main brush off
//         INWARD to turn the main brush on at full power in the inward direction
//         OUTWARD to turn the main brush on at full power in the outward direction
//
// @param:  sideBrush controls the Roomba's side brush.
//      Valid values are:
//         OFF to turn the side brush off
//         CCW to turn the side brush on at full power in the counter-clockwise direction
//         CW to turn the side brush on at full power in the clockwise direction
//
// @param:  vacuum controls the Roomba's vacuum.
//      Valid values are:
//         OFF to turn the vacuum off
//         ON to turn the vacuum on at full power
//      Note that the vacuum can only run in the forward direction.
//
// @return: None.
//
//*****************************************************************************
extern void pwmMotors (int8_t mainBrushDutyCycle, int8_t sideBrushDutyCycle, uint8_t
vacuumDutyCycle);
//*****************************************************************************
//
// @brief:  This command lets you control the speed of Roomba's main brush, side brush,
//         and vacuum independently.
//
// @param:  mainBrushDutyCycle controls the duty cycle (%) of the Roomba's main brush.
//      Valid values are -100 to 100 inclusive. Positive values turn the main brush
//      inward, while negative values turn the main brush outward.
//
// @param:  sideBrushDutyCycle controls the duty cycle (%) of the Roomba's side brush.
//      Valid values are -100 to 100 inclusive. Positive values turn the side brush
//      in the counter-clockwise direction, while negative values turn the side brush
//      in the clockwise direction.
//
// @param:  vacuumDutyCycle controls the duty cycle (%) of the Roomba's vacuum.
//      Valid values are 0 to 100 inclusive. The vacuum only runs in the forward
//      direction.
//
// @return: None.
//
//*****************************************************************************
extern void leds (bool checkRobot, bool home, bool spot, bool debris, uint8_t powerColor, uint8_t
powerIntensity);
//*****************************************************************************
//
// @brief:  This command controls the LEDs common to all models of Roomba 600.
//
// @param:  checkRobot controls the power state of the Check Robot LED, which is orange.
//      Valid values are false (to turn off the Check Robot LED) or true (to turn on
//      the Check Robot LED).
```

```
//
// @param:  home controls the power state of the Home / Dock LED, which is green.
//      Valid values are false (to turn off the Home / Dock LED) or true (to turn on
//      the Home / Dock LED).
//
// @param:  spot controls the power state of the Spot LED, which is green.
//      Valid values are false (to turn off the Spot LED) or true (to turn on
//      the Spot LED).
//
// @param:  debris controls the power state of the Debris LED, which is blue.
//      Valid values are false (to turn off the Debris LED) or true (to turn on
//      the Debris LED).
//
// @param:  powerColor controls the color state of the Power LED, which is a
//          bicolor (red/green) LED.
//      Valid values are 0 to 255 inclusive, where 0 = green, 255 = red, and
//      intermediate values are intermediate colors (orange, yellow, etc).
//
// @param:  powerIntensity controls the brightness of the Power LED, which is a
//          bicolor (red/green) LED.
//      Valid values are 0 to 255 inclusive, where 0 = off, 255 = full intensity,
//      and intermediate values are intermediate intensities.
//
// @return: None.
//
//*****************************************************************************
extern void weekdayLeds (bool sun, bool mon, bool tues, bool wed, bool thur, bool fri, bool sat);
//*****************************************************************************
//
// @brief:  This command controls the state of the weekday LEDs present on the Roomba 560 and 570.
//          These LEDs are located above the four 7-segment displays.
//
// @param:  sun controls the red Sunday scheduling LED.
//      Valid values are false (to turn off the Sunday LED) or true (to turn on the Sunday LED).
//
// @param:  mon controls the red Monday scheduling LED.
//      Valid values are false (to turn off the Monday LED) or true (to turn on the Monday LED).
//
// @param:  tues controls the red Tuesday scheduling LED.
//      Valid values are false (to turn off the Tuesday LED) or true (to turn on the Tuesday LED).
//
// @param:  wed controls the red Wednesday scheduling LED.
//      Valid values are false (to turn off the Wednesday LED) or true (to turn on the Wednesday LED).
//
// @param:  thur controls the red Thursday scheduling LED.
//      Valid values are false (to turn off the Thursday LED) or true (to turn on the Thursday LED).
//
// @param:  fri controls the red Friday scheduling LED.
//      Valid values are false (to turn off the Friday LED) or true (to turn on the Friday LED).
//
// @param:  sat controls the red Saturday scheduling LED.
//      Valid values are false (to turn off the Saturday LED) or true (to turn on the Saturday LED).
//
// @return: None.
//
//*****************************************************************************
```

```
extern void schedulingLeds (bool scheduleLed, bool clockLed, bool amLed, bool pmLed, bool colonLed);
//****************************************************************************************
//
// @brief:  This command controls the state of the scheduling LEDs present on the Roomba 560 and 570.
//
// @param:  scheduleLed controls the red Schedule LED.
//      Valid values are false (to turn off the Schedule LED) or true (to turn on the Schedule LED).
//
// @param:  clockLed controls the red Clock LED.
//      Valid values are false (to turn off the Clock LED) or true (to turn on the Clock LED).
//
// @param:  amLed controls the red AM LED.
//      Valid values are false (to turn off the AM LED) or true (to turn on the AM LED).
//
// @param:  pmLed controls the red PM LED.
//      Valid values are false (to turn off the PM LED) or true (to turn on the PM LED).
//
// @param:  colonLed controls the red Colon (:) LED.
//      Valid values are false (to turn off the Colon LED) or true (to turn on the Colon LED).
//
// @return: None.
//
//****************************************************************************************
extern void digitLedsRaw (DIGIT_POSITION digit, bool a, bool b, bool c, bool d, bool e, bool f, bool g);
//****************************************************************************************
//
// @brief:  This command controls the four 7 segment red LED displays on the
//          Roomba 560 and 570. The seven segments are labeled as follows:
//
//              ___A___
//            |         |
//          F|           |B
//            |___G___|
//            |         |
//          E|           |C
//            |___D___|
//
//          Note: this function cannot be used in conjunction with the digitLedsAscii
//          function. This function will also store previous values written to the
//          four 7-segment displays via the digitLedsRaw function. This can be cleared
//          either via the digitLedsRaw function (by manually turning off all segments
//          for each digit) or via the digitLedsRawClear function.
//
// @param:  digit selects which 7 segment display is being written to.
//      Valid values are FAR_LEFT, MIDDLE_LEFT, MIDDLE_RIGHT, and FAR_RIGHT.
//
// @param:  a controls the 'A' LED segment on the selected digit.
//      Valid values are false (to turn off the 'A' segment) or true (to turn
//      on the 'A' segment).
//
// @param:  b controls the 'B' LED segment on the selected digit.
//      Valid values are false (to turn off the 'B' segment) or true (to turn
//      on the 'B' segment).
//
// @param:  c controls the 'C' LED segment on the selected digit.
//      Valid values are false (to turn off the 'C' segment) or true (to turn
```

```
//     on the 'C' segment).
//
// @param:  d controls the 'D' LED segment on the selected digit.
//     Valid values are false (to turn off the 'D' segment) or true (to turn
//     on the 'D' segment).
//
// @param:  e controls the 'E' LED segment on the selected digit.
//     Valid values are false (to turn off the 'E' segment) or true (to turn
//     on the 'E' segment).
//
// @param:  f controls the 'F' LED segment on the selected digit.
//     Valid values are false (to turn off the 'F' segment) or true (to turn
//     on the 'F' segment).
//
// @param:  g controls the 'G' LED segment on the selected digit.
//     Valid values are false (to turn off the 'G' segment) or true (to turn
//     on the 'G' segment).
//
// @return: None.
//
//*****************************************************************************
extern void buttons (BUTTON button);
//*****************************************************************************
//
// @brief:  If given an argument of the name of one of the 8 buttons on the Roomba,
//         this function digitally presses the respective button on the Roomba.
//
//   Bit  |   7  |  6   |  5  |   4  |   3  |   2  |   1  |   0
//   ----------------------------------------------------------------------
//   Value | Clock  |Schedule|  Day  |  Hour  | Minute |  Dock  |  Spot  |  Clean
//
// @param:  digit selects which 7 segment display is being written to.
//     Valid values are CLEAN, SPOT, DOCK, MINUTE, HOUR, DAY, SCHEDULE, CLOCK, and
//     ALL.
//
// @return: None.
//
//*****************************************************************************
extern void digitLedsAscii (DIGIT_POSITION digit);
//*****************************************************************************
//
// @brief:  This command will print one of the ASCII characters listed in the table
//         below on the 7 segment display given as an argument. Due to the
//         limitations of a 7 segment display, all characters are an approximation
//         and not all ASCII codes are listed.
//
// |Code|Display|Code |  Display  |  Code  | Display |  Code  |Display|
//   32            53        5         70, 102     F      86, 118  V
//   33    !       54        6         71, 103     G      87, 119  W
//   34    ”       55        7         72, 104     H      88, 120  X
//   35    #       56        8         73, 105     I      89, 121  Y
//   37    %       57        9         74, 106     J      90, 122  Z
//   38    &       58        :         75, 107     K      91, 40   [
//   39    ’       59        ;         76, 108     L      92       \
//   44    ,       60        ¡         77, 109     M       93, 41  ]
//   45    -       61        =         78, 110     N      94       ^
```

```
//   46   .        62       ¿       79, 111     O     95      _
//   47   /        63       ?       80, 112     P     96      ‘
//   48   0        65, 97   A       81, 113     Q     123     {
//   49   1        66, 98   B       82, 114     R     124     —
//   50   2        67, 99   C       83, 36, 115 S     125     }
//   51   3        68, 100  D       84, 116     T     126     ~
//   52   4        69, 101  E       85, 117     U
//
//
// @param:  button selects which button on the Roomba to press.
//      Valid values are FAR_LEFT, MIDDLE_LEFT, MIDDLE_RIGHT, and FAR_RIGHT.
//
// @return: None.
//
//*****************************************************************************
extern void song (uint8_t songNumber, uint8_t songLength, uint8_t songNotes[],
uint8_t songNotesDuration[]);
//*****************************************************************************
//
// @brief:  This command lets you write up to 4 songs to store in the Roomba's memory,
//          each of which can be up to 16 notes long. The play() command can be used
//          to play one of the songs stored in memory. The duration of each note
//          should be input in units of 1/64th of a second (for example, a note
//          meant to last one-half second should be given a duration of 32). Refer to
//          the chart below for song numbers that correspond to each note.
//
// |Number|Note|Frequency |Number | Note |Frequency|Number| Note | Frequency |
//  31     G      49.0    58     A#     233.1   85     C#     1108.8
//  32     G#     51.9    59     B      246.9   86     D      1174.7
//  33     A      55.0    60     C      261.6   87     D#     1244.5
//  34     A#     58.3    61     C#     277.2   88     E      1318.5
//  35     B      61.7    62     D      293.7   89     F      1396.9
//  36     C      65.4    63     D#     311.1   90     F#     1480.0
//  37     C#     69.3    64     E      329.6   91     G      1568.0
//  38     D      73.4    65     F      349.2   92     G#     1661.3
//  39     D#     77.8    66     F#     370.0   93     A      1760.0
//  40     E      82.4    67     G      392.0   94     A#     1864.7
//  41     F      87.3    68     G#     415.3   95     B      1975.6
//  42     F#     92.5    69     A      440.0   96     C      2093.1
//  43     G      98.0    70     A#     466.2   97     C#     2217.5
//  44     G#     103.8   71     B      493.9   98     D      2349.4
//  45     A      110.0   72     C      523.3   99     D#     2489.1
//  46     A#     116.5   73     C#     554.4   100    E      2637.1
//  47     B      123.5   74     D      587.3   101    F      2793.9
//  48     C      130.8   75     D#     622.3   102    F#     2960.0
//  49     C#     138.6   76     E      659.3   103    G      3136.0
//  50     D      146.8   77     F      698.5   104    G#     3322.5
//  51     D#     155.6   78     F#     740.0   105    A      3520.1
//  52     E      164.8   79     G      784.0   106    A#     3729.4
//  53     F      174.6   80     G#     830.6   107    B      3951.2
//  54     F#     185.0   81     A      880.0
//  55     G      196.0   82     A#     932.4
//  56     G#     207.7   83     B      987.8
//  57     A      220.0   84     C      1046.5
//
// @param:  songNumber selects which of the 4 songs the new sequence is to be stored
```

```
//         to.
//     Valid values are between 0-3 inclusive.
//
// @param:  songLength describes the number of notes in the new song.
//     Valid values are between 1-16 inclusive.
//
// @param:  songNotes[] is an array of unsigned 8-bit integers that contains the number
//         for each note given by the table above.
//     Valid values are between 31-107 inclusive.
//
// @param:  songNotesDuration[] is an array of unsigned 8-bit integers that contains the
//         duration of each note in units of 1/64th of a second.
//     Valid values are between 0-255 inclusive.
//
// @return: None.
//
//*****************************************************************************
extern void play (uint8_t songNumber);
//*****************************************************************************
//
// @brief:  This command will play one of the 4 songs stored in memory using the song()
//         function.
//
// @param:  songNumber gives the function the song number from the Roomba's memory to
//         play.
//     Valid values are between 0-3 inclusive.
//
// @return: None.
//
//*****************************************************************************
extern void sensors (uint8_t packetId);
//*****************************************************************************
//
// @brief:  This command requests one of the 58 sensor packets available with the OI.
//
//|Group Packet ID|Packet Size|Contains Packets|
//  0             26          7 - 26
//  1             10          7 - 16
//  2             6          17 - 20
//  3             10          21 - 26
//  4             14          27 - 34
//  5             12          35 - 42
//  6             52          7 - 42
//  100           80           7 - 58
//  101           28           43 - 58
//  106           12           46 - 51
//  107           9            54 - 58
//
// @param:  packetId gives the packet ID of the requested packet to the OI. If given
//         a value of 6, this function will return all 58 packets, whle a value of
//         0-5 will return specific subgroups of sensors.
//     Valid values are between 0-58 inclusive and 100-107 inclusive.
//
// @return: int16_t result.
//
//*****************************************************************************
```

```
extern void queryList (uint8_t packetId, uint8_t packetNumber);
//*****************************************************************************
//
// @brief:  This command requests a list of different sensor packets whose results are
//          returned once, as in the sensors() command. The packets are returned in
//          the order specified.
//
// @param:  packetId gives the packet ID of the requested packet to the OI. If given
//          a value of 6, this function will return all 58 packets, whle a value of
//          0-5 will return specific subgroups of sensors.
//      Valid values are between 0-58 inclusive and 100-107 inclusive.
//
// @param:  packetNumber gives the number of packets requested in the list.
//      Valid values are between 1-67 inclusive.
//
// @return: int16_t result.
//
//*****************************************************************************
extern void stream (uint8_t packetId, uint8_t packetNumber);
//*****************************************************************************
//
// @brief:  This command starts a stream containing a list of different sensor packets
//          whose results are returned every 15 ms, which is the rate Roomba uses to
//          update data. The packets are returned in the order specified. The packets
//          are returned in the format:
//          [19][N-bytes][Packet ID 1][Packet 1 data...][Packet ID 2][Packet 2 data...]
//          [Checksum]
//          The checksum is a 1-byte value that is the 8-bit complement of all the
//          bytes in the packet, meaning if you add all of the bytes in the packet,
//          including the checksum, the low byte of the result will be 0.
//
// @param:  packetId gives the packet ID of the requested packet to the OI. If given
//          a value of 6, this function will return all 58 packets, whle a value of
//          0-5 will return specific subgroups of sensors.
//      Valid values are between 0-58 inclusive and 100-107 inclusive.
//
// @param:  packetNumber gives the number of packets requested in the list.
//      Valid values are between 1-67 inclusive.
//
// @return: int16_t result.
//
//*****************************************************************************
extern void pauseResumeStream (bool toggleStream);
//*****************************************************************************
//
// @brief:  This command either pauses or resumes the data stream given in stream(),
//          without clearing the list of requested packets.
//
// @param:  toggleStream describes whether to pause or resume the stream, with 0
//          indicating a pause and a 1 indicating a resume.
//      Valid values are 0 and 1.
//
// @return: None.
//*****************************************************************************
```

APPENDIX B

SOURCE CODE LISTING

```
/*
 * create2_driver.c
 *
 *  Created on: August 21, 2019
 *      Author: Zachary Holloway and Sophie Soueid
 */
#include "ti/devices/msp432p4xx/inc/msp.h"
#include "create2_driver.h"
#include "stdio.h"
#include "string.h"
#include "driverlib.h"

/* roombaSchedule function defines */
#define DAY_ENABLE                  (1)
#define SUNDAY_HOUR                 (2)
#define SUNDAY_MINUTE               (3)
#define MONDAY_HOUR                 (4)
#define MONDAY_MINUTE               (5)
#define TUESDAY_HOUR                (6)
#define TUESDAY_MINUTE              (7)
#define WEDNESDAY_HOUR              (8)
#define WEDNESDAY_MINUTE            (9)
#define THURSDAY_HOUR               (10)
#define THURSDAY_MINUTE             (11)
#define FRIDAY_HOUR                 (12)
#define FRIDAY_MINUTE               (13)
#define SATURDAY_HOUR               (14)
#define SATURDAY_MINUTE             (15)

/* motors function defines */
#define MAIN_BRUSH_DIRECTION        (0b00010000)
#define SIDE_BRUSH_DIRECTION        (0b00001000)
#define MAIN_BRUSH_POWER            (0b00000100)
#define VACUUM_POWER                (0b00000010)
#define SIDE_BRUSH_POWER            (0b00000001)

/* leds function defines */
#define CHECK_ROBOT_LED             (0b00001000)
#define HOME_DOCK_LED               (0b00000100)
#define SPOT_LED                    (0b00000010)
#define DEBRIS_LED                  (0b00000001)

/* weekdayLeds function defines */
#define SUNDAY_LED                  (0b00000001)
#define MONDAY_LED                  (0b00000010)
#define TUESDAY_LED                 (0b00000100)
#define WEDNESDAY_LED               (0b00001000)
#define THURSDAY_LED                (0b00010000)
#define FRIDAY_LED                  (0b00100000)
#define SATURDAY_LED                (0b01000000)

/* scheduleLeds function defines */
#define SCHEDULE_LED                (0b00010000)
#define CLOCK_LED                   (0b00001000)
#define AM_LED                      (0b00000100)
#define PM_LED                      (0b00000010)
```

```
#define COLON_LED                   (0b00000001)

/* digitLedsRaw function defines */
#define RAW_LED_A                   (0b00000001)
#define RAW_LED_B                   (0b00000010)
#define RAW_LED_C                   (0b00000100)
#define RAW_LED_D                   (0b00001000)
#define RAW_LED_E                   (0b00010000)
#define RAW_LED_F                   (0b00100000)
#define RAW_LED_G                   (0b01000000)

#define EUSCI_A0_BASE                 (PERIPH_BASE +0x00001000)
#define EUSCI_A1_BASE                 (PERIPH_BASE +0x00001400)
#define EUSCI_A2_BASE                 (PERIPH_BASE +0x00001800)
#define EUSCI_A3_BASE                 (PERIPH_BASE +0x00001C00)
//#define EUSCI_A_UART_CLOCKSOURCE_SMCLK     (EUSCI_A_CTLW0_SSEL__SMCLK)
//#define EUSCI_A_UART_CLOCKSOURCE_ACLK      (EUSCI_A_CTLW0_SSEL__ACLK)

uint8_t commandBuffer[20];
int16_t responseBuffer[20];
int16_t rxBuffer[1024];
uint16_t roombaBufferIndex = 0;
uint8_t song0Array[34] = {CREATE2_OPCODE_SONG, 0};
uint8_t song1Array[34] = {CREATE2_OPCODE_SONG, 1};
uint8_t song2Array[34] = {CREATE2_OPCODE_SONG, 2};
uint8_t song3Array[34] = {CREATE2_OPCODE_SONG, 3};
uint8_t scheduleArray[] = {CREATE2_OPCODE_SCHEDULE, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
uint8_t schedulingLedsArray[] = {CREATE2_OPCODE_SCHEDULING_LEDS, 0, 0};
uint8_t digitLedsRawArray[] = {CREATE2_OPCODE_DIGIT_LEDS_RAW, 0, 0, 0, 0};
uint8_t digitLedsAsciiArray[] = {CREATE2_OPCODE_DIGIT_LEDS_ASCII, 0, 0, 0, 0};
uint8_t packetIDList[59] = {CREATE2_OPCODE_SENSORS};

uint32_t clockSpeed;
uint32_t eusciChannel;
uint8_t eusciFlag;

void sendUART (void) {
   int i;
   int length = sizeof(commandBuffer);
   for (i = 0; i < length; i++){
      while (!(EUSCI_A2->IFG & EUSCI_A_IFG_TXIFG));
      EUSCI_A2->TXBUF = commandBuffer[i];
   }
   memset(commandBuffer, 0, sizeof(commandBuffer));
}

void roombaStart (void) {
   commandBuffer[0] = CREATE2_OPCODE_START;
   sendUART();
   memset(commandBuffer, 0, sizeof(commandBuffer));
}

void roombaReset (void) {
   commandBuffer[0] = CREATE2_OPCODE_RESET;
   sendUART();
   memset(commandBuffer, 0, sizeof(commandBuffer));
```

```
}

void roombaStop (void) {
   commandBuffer[0] = CREATE2_OPCODE_STOP;
   sendUART();
   memset(commandBuffer, 0, sizeof(commandBuffer));
}

void setRoombaBaud (uint32_t valueBaud) {

   switch (valueBaud)
   {
      case 300:
         commandBuffer[0] = CREATE2_OPCODE_BAUD;
         commandBuffer[1] = 0;
         break;
      case 600:
         commandBuffer[0] = CREATE2_OPCODE_BAUD;
         commandBuffer[1] = 1;
         break;
      case 1200:
         commandBuffer[0] = CREATE2_OPCODE_BAUD;
         commandBuffer[1] = 2;
         break;
      case 2400:
         commandBuffer[0] = CREATE2_OPCODE_BAUD;
         commandBuffer[1] = 3;
         break;
      case 4800:
         commandBuffer[0] = CREATE2_OPCODE_BAUD;
         commandBuffer[1] = 4;
         break;
      case 9600:
         commandBuffer[0] = CREATE2_OPCODE_BAUD;
         commandBuffer[1] = 5;
         break;
      case 14400:
         commandBuffer[0] = CREATE2_OPCODE_BAUD;
         commandBuffer[1] = 6;
         break;
      case 19200:
         commandBuffer[0] = CREATE2_OPCODE_BAUD;
         commandBuffer[1] = 7;
         break;
      case 28800:
         commandBuffer[0] = CREATE2_OPCODE_BAUD;
         commandBuffer[1] = 8;
         break;
      case 38400:
         commandBuffer[0] = CREATE2_OPCODE_BAUD;
         commandBuffer[1] = 9;
         break;
      case 57600:
         commandBuffer[0] = CREATE2_OPCODE_BAUD;
         commandBuffer[1] = 10;
         break;
```

```
        case 115200:
            commandBuffer[0] = CREATE2_OPCODE_BAUD;
            commandBuffer[1] = 11;
            break;
        case default:
            return;
    }
    sendUART();
    memset(commandBuffer, 0, sizeof(commandBuffer));

    //delay 100ms
}

void roombaSafe (void) {
    commandBuffer[0] = CREATE2_OPCODE_SAFE;
    sendUART();
    memset(commandBuffer, 0, sizeof(commandBuffer));
}

void roombaFull (void) {
    commandBuffer[0] = CREATE2_OPCODE_FULL;
    sendUART();
    memset(commandBuffer, 0, sizeof(commandBuffer));
}

void roombaClean (void) {
    commandBuffer[0] = CREATE2_OPCODE_CLEAN;
    sendUART();
    memset(commandBuffer, 0, sizeof(commandBuffer));
}

void roombaMax (void) {
    commandBuffer[0] = CREATE2_OPCODE_MAX;
    sendUART();
    memset(commandBuffer, 0, sizeof(commandBuffer));
}

void roombaSpot (void) {
    commandBuffer[0] = CREATE2_OPCODE_SPOT;
    sendUART();
    memset(commandBuffer, 0, sizeof(commandBuffer));
}

void roombaSeekDock (void) {
    commandBuffer[0] = CREATE2_OPCODE_SEEK_DOCK;
    sendUART();
    memset(commandBuffer, 0, sizeof(commandBuffer));
}

void roombaPower (void) {
    commandBuffer[0] = CREATE2_OPCODE_POWER;
    sendUART();
    memset(commandBuffer, 0, sizeof(commandBuffer));
}

void roombaSchedule (DAY_VALUE day, bool enable, uint8_t hour, uint8_t minute)
```

```
{
   if ((hour > 23)||(minute > 59))
      return;
   else if (enable) //if enable is true
      scheduleArray[DAY_ENABLE] |= day; //set the day enable value
   else           //if enable is false
      scheduleArray[DAY_ENABLE] &= ~day; //clear the day enable value

   switch (day)
   {
      case SUNDAY:
         scheduleArray[SUNDAY_HOUR] = hour;
         scheduleArray[SUNDAY_MINUTE] = minute;
         break;
      case MONDAY:
         scheduleArray[MONDAY_HOUR] = hour;
         scheduleArray[MONDAY_MINUTE] = minute;
         break;
      case TUESDAY:
         scheduleArray[TUESDAY_HOUR] = hour;
         scheduleArray[TUESDAY_MINUTE] = minute;
         break;
      case WEDNESDAY:
         scheduleArray[WEDNESDAY_HOUR] = hour;
         scheduleArray[WEDNESDAY_MINUTE] = minute;
         break;
      case THURSDAY:
         scheduleArray[THURSDAY_HOUR] = hour;
         scheduleArray[THURSDAY_MINUTE] = minute;
         break;
      case FRIDAY:
         scheduleArray[FRIDAY_HOUR] = hour;
         scheduleArray[FRIDAY_MINUTE] = minute;
         break;
      case SATURDAY:
         scheduleArray[SATURDAY_HOUR] = hour;
         scheduleArray[SATURDAY_MINUTE] = minute;
         break;
   }

   memcpy(commandBuffer, scheduleArray, sizeof(scheduleArray));
   sendUART();
   memset(commandBuffer, 0, sizeof(commandBuffer));
}

void roombaScheduleDisableAll (void) {
   memset(scheduleArray, 0, sizeof(scheduleArray));
   scheduleArray[0] = CREATE2_OPCODE_SCHEDULE;
   memcpy(commandBuffer, scheduleArray, sizeof(scheduleArray));
   sendUART();
   memset(commandBuffer, 0, sizeof(commandBuffer));
}

void roombaSetDayTime (DAY_VALUE day, uint8_t hour, uint8_t minute) {
   commandBuffer[0] = CREATE2_OPCODE_SET_DAY_TIME;
   commandBuffer[1] = day;
```

```
    commandBuffer[2] = hour;
    commandBuffer[3] = minute;
    sendUART();
    memset(commandBuffer, 0, sizeof(commandBuffer));
}

void roombaDrive (int16_t velocity, int16_t radius) {
    commandBuffer[0] = CREATE2_OPCODE_DRIVE;
    commandBuffer[1] = velocity >> 8;
    commandBuffer[2] = velocity & 0x00FF;
    commandBuffer[3] = radius >> 8;
    commandBuffer[4] = radius & 0x00FF;
    sendUART();
    memset(commandBuffer, 0, sizeof(commandBuffer));
}

void roombaDriveDirect (int16_t rightVelocity, int16_t leftVelocity) {
    commandBuffer[0] = CREATE2_OPCODE_DRIVE_DIRECT;
    commandBuffer[1] = rightVelocity >> 8;
    commandBuffer[2] = rightVelocity & 0x00FF;
    commandBuffer[3] = leftVelocity >> 8;
    commandBuffer[4] = leftVelocity & 0x00FF;
    sendUART();
    memset(commandBuffer, 0, sizeof(commandBuffer));
}

void roombaDrivePWM (int16_t rightDutyCycle, int16_t leftDutyCycle) {
    int16_t rightPWM;
    int16_t leftPWM;
    rightPWM = 255*rightDutyCycle/100;
    leftPWM = 255*leftDutyCycle/100;

    commandBuffer[0] = CREATE2_OPCODE_DRIVE_PWM;
    commandBuffer[1] = rightPWM >> 8;
    commandBuffer[2] = rightPWM & 0x00FF;
    commandBuffer[3] = leftPWM >> 8;
    commandBuffer[4] = leftPWM & 0x00FF;
    sendUART();
    memset(commandBuffer, 0, sizeof(commandBuffer));
}

void motors (MOTORS_MAIN_BRUSH mainBrush, MOTORS_SIDE_BRUSH sideBrush,
MOTORS_VACUUM vacuum) {
    uint8_t motorsControl;
    motorsControl = 0;

    switch (mainBrush)
    {
        case OFF:
            motorsControl &= ~MAIN_BRUSH_POWER; // turn off main brush
            break;
        case INWARD:
            motorsControl |= MAIN_BRUSH_POWER; // turn on main brush
            motorsControl &= ~MAIN_BRUSH_DIRECTION; // set main brush to inward
            break;
        case OUTWARD:
```

```
        motorsControl |= MAIN_BRUSH_POWER; // turn on main brush
        motorsControl |= MAIN_BRUSH_DIRECTION; // set main brush to outward
        break;
    }

    switch (sideBrush)
    {
      case OFF:
        motorsControl &= ~SIDE_BRUSH_POWER; // turn off side brush
        break;
      case CCW:
        motorsControl |= SIDE_BRUSH_POWER; // turn on side brush
        motorsControl &= ~SIDE_BRUSH_DIRECTION; // set side brush to ccw
        break;
      case CW:
        motorsControl |= SIDE_BRUSH_POWER; // turn on side brush
        motorsControl |= SIDE_BRUSH_DIRECTION; // set side brush to cw
        break;
    }

    switch (vacuum)
    {
      case OFF:
        motorsControl &= ~VACUUM_POWER; // turn off vacuum
        break;
      case ON:
        motorsControl |= VACUUM_POWER; // turn on vacuum
        break;
    }

    commandBuffer[0] = CREATE2_OPCODE_MOTORS;
    commandBuffer[1] = motorsControl;
    sendUART();
    memset(commandBuffer, 0, sizeof(commandBuffer));
}

void pwmMotors (int8_t mainBrushDutyCycle, int8_t sideBrushDutyCycle, uint8_t vacuumDutyCycle) {
    mainBrushDutyCycle = 127*mainBrushDutyCycle/100;
    sideBrushDutyCycle = 127*sideBrushDutyCycle/100;
    vacuumDutyCycle = 127*vacuumDutyCycle/100;

    commandBuffer[0] = CREATE2_OPCODE_PWM_MOTORS;
    commandBuffer[1] = mainBrushDutyCycle;
    commandBuffer[2] = sideBrushDutyCycle;
    commandBuffer[3] = vacuumDutyCycle;
    sendUART();
    memset(commandBuffer, 0, sizeof(commandBuffer));
}

void leds (bool checkRobot, bool home, bool spot, bool debris, uint8_t powerColor, uint8_t
powerIntensity) {
    uint8_t ledBits;
    ledBits = 0;

    if (checkRobot)
        ledBits |= CHECK_ROBOT_LED;
```

```
    else
      ledBits &= ~CHECK_ROBOT_LED;

    if (home)
      ledBits |= HOME_DOCK_LED;
    else
      ledBits &= ~HOME_DOCK_LED;

    if (spot)
      ledBits |= SPOT_LED;
    else
      ledBits &= ~SPOT_LED;

    if (debris)
      ledBits |= DEBRIS_LED;
    else
      ledBits &= ~DEBRIS_LED;

    commandBuffer[0] = CREATE2_OPCODE_LEDS;
    commandBuffer[1] = ledBits;
    commandBuffer[2] = powerColor;
    commandBuffer[3] = powerIntensity;
    sendUART();
    memset(commandBuffer, 0, sizeof(commandBuffer));
}

void weekdayLeds (bool sun, bool mon, bool tues, bool wed, bool thur, bool fri, bool sat) {

    if (sun)
      schedulingLedsArray[1] |= SUNDAY_LED;
    else
      schedulingLedsArray[1] &= ~SUNDAY_LED;

    if (mon)
      schedulingLedsArray[1] |= MONDAY_LED;
    else
      schedulingLedsArray[1] &= ~MONDAY_LED;

    if (tues)
      schedulingLedsArray[1] |= TUESDAY_LED;
    else
      schedulingLedsArray[1] &= ~TUESDAY_LED;

    if (wed)
      schedulingLedsArray[1] |= WEDNESDAY_LED;
    else
      schedulingLedsArray[1] &= ~WEDNESDAY_LED;

    if (thur)
      schedulingLedsArray[1] |= THURSDAY_LED;
    else
      schedulingLedsArray[1] &= ~THURSDAY_LED;

    if (fri)
      schedulingLedsArray[1] |= FRIDAY_LED;
    else
```

```
    schedulingLedsArray[1] &= ~FRIDAY_LED;

  if (sat)
    schedulingLedsArray[1] |= SATURDAY_LED;
  else
    schedulingLedsArray[1] &= ~SATURDAY_LED;

  memcpy(commandBuffer, schedulingLedsArray, sizeof(schedulingLedsArray));
  sendUART();
  memset(commandBuffer, 0, sizeof(commandBuffer));
}

void schedulingLeds (bool scheduleLed, bool clockLed, bool amLed, bool pmLed, bool colonLed) {

  if (scheduleLed)
    schedulingLedsArray[2] |= SCHEDULE_LED;
  else
    schedulingLedsArray[2] &= ~SCHEDULE_LED;

  if (clockLed)
    schedulingLedsArray[2] |= CLOCK_LED;
  else
    schedulingLedsArray[2] &= ~CLOCK_LED;

  if (amLed)
    schedulingLedsArray[2] |= AM_LED;
  else
    schedulingLedsArray[2] &= ~AM_LED;

  if (pmLed)
    schedulingLedsArray[2] |= PM_LED;
  else
    schedulingLedsArray[2] &= ~PM_LED;

  if (colonLed)
    schedulingLedsArray[2] |= COLON_LED;
  else
    schedulingLedsArray[2] &= ~COLON_LED;

  memcpy(commandBuffer, schedulingLedsArray, sizeof(schedulingLedsArray));
  sendUART();
  memset(commandBuffer, 0, sizeof(commandBuffer));
}

void digitLedsRaw (DIGIT_POSITION digit, bool a, bool b, bool c, bool d, bool e, bool f, bool g) {

  if (a)
    digitLedsRawArray[digit] |= RAW_LED_A;
  else
    digitLedsRawArray[digit] &= ~RAW_LED_A;

  if (b)
    digitLedsRawArray[digit] |= RAW_LED_B;
  else
    digitLedsRawArray[digit] &= ~RAW_LED_B;
```

```
    if (c)
        digitLedsRawArray[digit] |= RAW_LED_C;
    else
        digitLedsRawArray[digit] &= ~RAW_LED_C;

    if (d)
        digitLedsRawArray[digit] |= RAW_LED_D;
    else
        digitLedsRawArray[digit] &= ~RAW_LED_D;

    if (e)
        digitLedsRawArray[digit] |= RAW_LED_E;
    else
        digitLedsRawArray[digit] &= ~RAW_LED_E;

    if (f)
        digitLedsRawArray[digit] |= RAW_LED_F;
    else
        digitLedsRawArray[digit] &= ~RAW_LED_F;

    if (g)
        digitLedsRawArray[digit] |= RAW_LED_G;
    else
        digitLedsRawArray[digit] &= ~RAW_LED_G;


    memcpy(commandBuffer, digitLedsRawArray, sizeof(digitLedsRawArray));
    sendUART();
    memset(commandBuffer, 0, sizeof(commandBuffer));
}

void buttons (BUTTON button) {
    commandBuffer[0] = CREATE2_OPCODE_BUTTONS;
    commandBuffer[1] = button;
    sendUART();
    memset(commandBuffer, 0, sizeof(commandBuffer));
}

void digitLedsAscii (DIGIT_POSITION digit) {
    char string = " !\"#%&',-./0123456789:;¡=¿?ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]^_`{|}~";
        //these are all the characters that are possible for display
    char ascii;
    char *returnPointer;

    ch = digitLedsRawArray[digit];
    ret = strchr(str, ascii);
    if (!returnPointer)
        print("Error: Character is not valid for this display.\n")
    else
        commandBuffer[digit] = ascii;

    memcpy(commandBuffer, digitLedsAsciiArray, sizeof(digitLedsAsciiArray));
    sendUART();
    memset(commandBuffer, 0, sizeof(commandBuffer));
}
```

```
void song (uint8_t songNumber, uint8_t songLength, uint8_t songNotes[], uint8_t songNotesDuration[]) {
   uint8_t i;

   if (songNumber == 0) {
      for (i = 0; i < songLength; i++) {
         song0Array[2 * i + 2] = songNotes[i];
         song0Array[2 * i + 3] = songNotesDuration[i];
         memcpy(commandBuffer, song0Array, sizeof(song0Array));
      }
   }
   else if (songNumber == 1) {
      for (i = 0; i < songLength; i++) {
         song1Array[2 * i + 2] = songNotes[i];
         song1Array[2 * i + 3] = songNotesDuration[i];
         memcpy(commandBuffer, song1Array, sizeof(song0Array));
      }
   }
   else if (songNumber == 2) {
      for (i = 0; i < songLength; i++) {
         song2Array[2 * i + 2] = songNotes[i];
         song2Array[2 * i + 3] = songNotesDuration[i];
         memcpy(commandBuffer, song2Array, sizeof(song0Array));
      }
   }
   else if (songNumber == 3) {
      for (i = 0; i < songLength; i++) {
         song3Array[2 * i + 2] = songNotes[i];
         song3Array[2 * i + 3] = songNotesDuration[i];
         memcpy(commandBuffer, song3Array, sizeof(song0Array));
      }
   }

   sendUART();
   memset(commandBuffer, 0, sizeof(commandBuffer));
}

void play (uint8_t songNumber) {
   commandBuffer[0] = CREATE2_OPCODE_PLAY;
   commandBuffer[1] = songNumber;
   sendUART();
   memset(commandBuffer, 0, sizeof(commandBuffer));
}
 void sensors (uint8_t packetId) {
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = packetId;
    sendUART();
    memset(commandBuffer, 0, sizeof(commandBuffer));
 }

 void queryList (uint8_t packetIdList[], uint8_t packetNumber) {
    uint8_t i;
    commandBuffer[0] = CREATE2_OPCODE_QUERY_LIST;
    commandBuffer[1] = packetNumber;
    for (i = 0; i < 58; i++)
       commandBuffer[i+2] = packetIdList[i];
    sendUART();
```

```
    memset(commandBuffer, 0, sizeof(commandBuffer));
}

 void stream (uint8_t packetIdList[], uint8_t packetNumber) {
    uint8_t i;
    commandBuffer[0] = CREATE2_OPCODE_STREAM;
    commandBuffer[1] = packetNumber;
    for (i = 0; i < 58; i++)
        commandBuffer[i+2] = packetIdList[i];
    sendUART();
    memset(commandBuffer, 0, sizeof(commandBuffer));
}

void pauseResumeStream(bool toggleStream) {
    commandBuffer[0] = CREATE2_OPCODE_PAUSE_RESUME_STREAM;
    commandBuffer[1] = toggleStream;
    sendUART();
    memset(commandBuffer, 0, sizeof(commandBuffer));
}
```

REFERENCES

Hackster.io, “Driving the iRobot Create 2 with the MSP432P401R”. [Online]. Available: https://www.hackster.io/holloway-soueid/driving-the-irobot-create-2-with-the-msp432p401r-35fc16

iRobot, “iRobot® Create® 2 Open Interface (OI) Specification based on the iRobot® Roomba® 600,” Adafruit. [Online]. Available: https://cdn-shop.adafruit.com/datasheets/create_2_Open_Interface_Spec.pdf.

BIOGRAPHICAL INFORMATION

Sophie Soueid is a senior at the University of Texas at Arlington who is pursuing both a B.S. in Electrical Engineering and a B.A. in French. She also has minors in Mathematics, French Localization & Translation, and Business Administration, and is currently taking graduate courses in electrical engineering through the fast-track program. During her time at UTA, Sophie has served as an officer in five different student organizations, the engineering ones being Eta Kappa Nu and Tau Beta Pi, and was named the awardee of the Dr. Wayne Duke Outstanding Student Leader Award in Spring 2019.

In addition to this driver and her senior design project, Sophie has worked on several projects in the field of Electrical Engineering, including an REU project to create a wearable magnetic sensor with tactile feedback, a junior design project to create a heart rate alarm clock, an undergraduate course project to create a multimeter, and a graduate course project to create a programmable pulse generator. In regard to her French degree, Sophie participated in the UTA in Montreal study abroad program in Summer 2018 and also plans to obtain Honors with a thesis focused on an exploration of machine translation of a low-resource French variant.

Sophie is now finishing up her third semester as a TA for Circuit Analysis I and her first year as a CWEP for Lockheed Martin. Immediately after graduating, she plans to start a full-time position in Component Engineering at Lockheed Martin Missiles and Fire Control and plans to concurrently to pursue an M.S. in Electrical Engineering part-time.