5-1-2020

DRIVING THE IROBOT CREATE2 – DATA FUNCTIONS

Zachary Holloway

DRIVING THE IROBOT CREATE2 – DATA FUNCTIONS

by

ZACHARY HOLLOWAY

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

I would like to start off by acknowledging by faculty mentor, Dr. Greg Turner. During this project and through several before, Dr. Turner has guided me eagerly and always had trust in me that I would get things done the right way. I immensely appreciate his guidance and would not have had half the opportunities I have been given through the electrical engineering department without his support. I would also like to acknowledge my colleague and friend who developed the other half of this driver, Sophie Soueid. She is a wonderful colleague to work with and kept my work neat and in check.

I would also like to acknowledge the friends I have found through the IEEE-Eta Kappa Nu organization. IEEE is a home away from home for me, and no matter when I walk into the mentoring office, I know that I will always have a friend waiting for me there. I would not have stayed motivated to work and learn without their laughter, determination, and support.

Lastly, I would like to thank my family. My parents raised me to always work hard at any task or study I take on and have given me all the tools needed to survive in the world that a college education cannot give you. I cannot possibly thank them, and the rest of my family, enough for everything they have done for me over the years.

December 10, 2019

ABSTRACT

DRIVING THE IROBOT CREATE2 – DATA FUNCTIONS

Zachary Holloway, B.S. Electrical Engineering

The University of Texas at Arlington, 2020

Faculty Mentor: Greg Turner

A driver was developed for the MSP432P401R microcontroller and the iRobot Create 2's data packets in the C programming language. The driver enables less experienced programmers to easily interface between these two devices without having to directly send, receive, and interpret byte data packets. This was developed using the Open Interface provided by iRobot that describes each of the Create 2's data packets, and the meaning of any data returned to the microcontroller. All communication between the two devices utilizes Universal Asynchronous Receiver/Transmitter, or UART. The driver is easily incorporated into any C project and provides functions that provides the full functionality of the Create 2. This could easily be altered to accommodate interface with any microcontroller with only minor modifications.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Motivation

Microcontrollers have started to become more mainstream with amateur programmers, as evidenced by the recent rise in popularity of hobbyist microcontrollers. However, communication protocols such as UART remain a large roadblock for hobbyist users, as it can be quite difficult to parse data that is sent in byte form without the prerequisite knowledge of electrical engineering. In response to this, a driver was developed to interface between the iRobot Create 2 and the MSP432P401R that implements functions that interpret the returned data from the data packet requests and present it in a way that is more meaningful to beginner and amateur programmers, while still retaining full functionality of all possible data packets that can be requested from the iRobot Create 2. This portion of the driver was specific to the sensor packet functions, meaning the functions that did not instruct the iRobot Create2 to perform any action, and instead only return data stored on the microcontroller onboard the Create2 itself.

1.2 Origin of the Driver

This specific project stemmed from an electrical engineering senior design project sponsored by Dr. Greg Turner, known as the Pet Waste Avoiding Autonomous Vacuum (PWAAV). This project specifically required the use of the iRobot Create2 and the MSP432P401R microcontroller, as well as the Pixy2 camera, to recognize and actively avoid detected liquid and solid pet waste. In order to accomplish this, the MSP432 needed

to be able to communicate with the Create2 to command it to avoid detected pet waste. In addition to this driver being required in order to create easy and clean programming possible for this project, it also enables other hobbyists, which is the Create2's primary marketing target, to create projects using the Create2 without requiring the education of a Bachelor's degree in electrical engineering.

CHAPTER 2

BREAKDOWN OF DATA FUNCTIONS

2.1 Overview

The driver itself consists of a header, or .h file, and a main source file, or .c file. The header file contains the definitions of all opcodes and packet IDs used to send commands and request data from the Create2. In addition, it also contains external prototype definitions for every function used in the main source file. Each of the data functions consists of four parts: preparing opcodes and packet IDs to send, sending them over UART, receiving data over UART, and parsing it before returning it to the user. When a data function is first called, the opcode for sensor data request is loaded into the command buffer, which is an array that stores 8-bit unsigned integers. The sendUART function pulls each 8-bit unsigned integer and loads it into the transmit first-in first-out (FIFO) buffer and begins to send those bytes along the data lines to the microcontroller onboard the Create2. The MSP432 then waits to for the receive line, or RX line, flag to be set high, indicating that the MSP432 is receiving data from the Create2. After waiting for the data transfer to finish, the function will then look at the data left in the receive buffer and parse it into data meaningful to the user before it is returned.

2.2 Preparing Transmit Data

The process begins by preparing bytes to be sent to the Create2's onboard microcontroller. In the driver, all opcodes and data packet IDs are a predetermined byte value, which is available in the Create2's Open Interface document. Every one of these

values has been predefined by what is known as a define statement, which assigns a value to a name that can be used in its place. For example, instead of having to load the value 8 into the command buffer to indicate the packet ID for the wall sensor, the driver instead fills it with the value CREATE2_PACKET_WALL. By doing this, the driver becomes significantly more readable for any end user who needs to understand how the driver works in order to suit it to their purposes. For all data functions, the byte preparation step begins by setting the first byte in the command buffer as CREATE2_OPCODE_SENSORS. This is a define statement that represents the opcode for a command that indicates to the Create2 that the MSP432 is requesting data, and that the next byte sent will be a packet ID that will indicate which specific sensor onboard the Create2 the MSP432 would like to pull data from. Then, the packet ID is loaded into the command buffer, and this packet ID will change from function to function. Once these two bytes are set in the command buffer, the user calls the function sendUART in order to begin the process of transmitting data from the MSP432 to the Create2.

2.3 Sending and Receiving Data

In order to send this data to the Create2, the sendUART function must load each of the 8-bit values stored in the array byte by byte. It runs code in a for loop for a given number of iterations that progressively takes each byte in the command buffer array and loads it into the transmit FIFO buffer of the MSP432. Once data is detected in the FIFO buffer, the UART module onboard the MSP432 automatically begins the transmit process at a baud rate of 115200, which is the default baud rate supported by the Create2.

After all data has been sent, the MSP432 sits idle as it waits for data to be received. The execution of code is halted by making it wait for a boolean value to be set to true before

it can advance in the code. This boolean is set to be true in an the UART receive interrupt, which is a function that is not called but instead triggered by an outside event. In this case, the code enters the interrupt when data is detected coming on the receiving line of the MSP432, and it begins to take this data and store it byte by byte into a receive buffer. After the first byte is received, the boolean is set to true and exits the interrupt. Once normal code execution resumes, the code waits for the receive flag to be set low, indicating that the data transfer is complete. Once this flag is finally set low, it can begin to look at the contents of the receive buffer and parse it.

2.4 Parsing Received Data

The parsing of data is the final critical step, as without parsing the data, the information returned would be virtually meaningless to anyone without a strong understanding of digital systems, including programming variable types and binary strings.

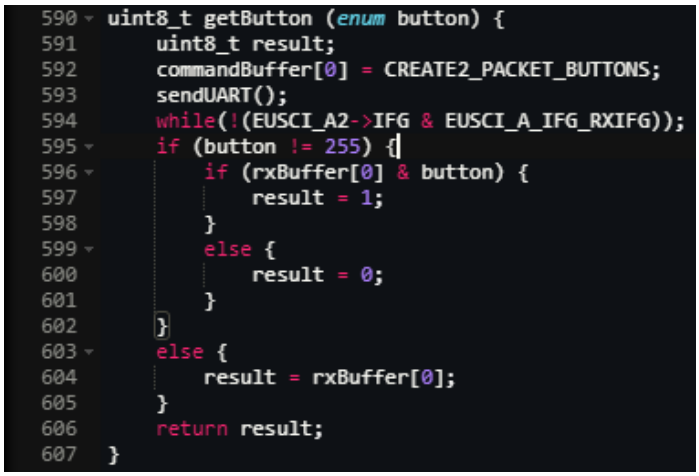
Buttons

Bit	7	6	5	4	3	2	1	0
Value	Clock	Schedule	Day	Hour	Minute	Dock	Spot	Clean

Table 2.1: Table of Button Binary String

As an example, the function “getButton” returns a binary string with possible values that range between 0 and 255 inclusive when converted into decimal. Without an understanding of binary strings, which many hobbyist programmers may lack, this will not be at all useful or meaningful. What this data actually means is that and 8-bit value is returned from the Create2, each of which can have a value of 0 or 1, and each bit in the string is representative of a button. If the button is pressed, the value of the corresponding bit is a 1, and if the button is unpressed, the corresponding bit is a 0. This means that the

8-bit string returned from the getButton function is representative of the 8 buttons located on the top of the Create2. In order to make this more understandable for the end user, the getButton function is a function that is of type enum, meaning that it only accepts a number of predefined enumerated arguments, each of which correspond to a certain integer value. Each button onboard the Create2 is assigned a name (i.e. “SCHEDULE”) that corresponds to its decimal value if the bit it is represented by in the binary string was a 1, and all other bits were a 0. As an example, the schedule button is represented by bit 6 in the binary string, so the decimal value that represents this is 2^6, which is equivalent to 64.



```
590 uint8_t getButton (enum button) {
591     uint8_t result;
592     commandBuffer[0] = CREATE2_PACKET_BUTTONS;
593     sendUART();
594     while(!(EUSCI_A2->IFG & EUSCI_A_IFG_RXIFG));
595     if (button != 255) {
596         if (rxBuffer[0] & button) {
597             result = 1;
598         }
599         else {
600             result = 0;
601         }
602     }
603     else {
604         result = rxBuffer[0];
605     }
606     return result;
607 }
```

Figure 2.1: Image of Button Data Function

Then, when the data is returned from the Create2, a logical OR operation is performed between the value in the receive buffer and the integer value assigned to the enum. This results in a binary value that is either a 1 or a 0, with 1 meaning that the button the user specified in the function was pressed, and 0 meaning this button was unpressed. This, however, comes with the tradeoff of only being able to check one button at a time. In order to counter this, there is an option in the enum called ALL, which will simply return the entire binary string and allow the end user to interpret the data from this string themselves, allowing them to receive data about all 8 buttons onboard the Create2 at the

same time. This type of data parsing is used in several functions that make up the Create2 data functions. In this way, the data function portion of the driver greatly simplifies the experience for the end user while still retaining full functionality for more experienced programmers.

CHAPTER 3

CONCLUSION

The data function portion of this driver opens up a great deal of possibilities for both hobbyist and experienced programmers alike when using the MSP432 and the Create2. The uses for this driver in hobbyist and electrical engineering projects are nearly endless when combined with the command functions portion of the driver. In addition, it would also be possible to add more features onto the existing senior design project this driver originated from, such as using the data functions to help build a complete map of the room or house the Create2 is cleaning for the end user to view. Finally, as the driver will be published as open source, it could be easily modified for anyone to use any microcontroller capable of UART communication to greatly expand the availability of the Create2's functionality.

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_SENSORS                  (142)
#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_VELOCITY        (41)
#define CREATE2_PACKET_REQUESTED_LEFT_VELOCITY         (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 {
  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 uint8_t getWheelDropLeft(void);
//****************************************************************************************
//
// @brief:  This command will return an integer (1 or 0) that indicates whether the
//          left wheel has dropped off a platform.
//
//
// @return: uint8_t result.
//
```

```
//****************************************************************************
extern uint8_t getWheelDropRight(void);
//****************************************************************************
//
// @brief:  This command will return an integer (1 or 0) that indicates whether the
//          right wheel has dropped off a platform.
//
//
// @return: uint8_t result.
//
//****************************************************************************
extern uint8_t getBumpLeft(void);
//****************************************************************************
//
// @brief:  This command will return an integer (1 or 0) that indicates whether the
//          left bump sensor has been triggered.
//
//
// @return: uint8_t result.
//
//****************************************************************************
extern uint8_t getBumpRight(void);
//****************************************************************************
//
// @brief:  This command will return an integer (1 or 0) that indicates whether the
//          right bump sensor has been triggered.
//
//
// @return: uint8_t result.
//
//****************************************************************************
extern uint8_t getWall(void);
//****************************************************************************
//
// @brief:  This command will return an integer (1 or 0) that indicates whether the
//          Roomba detects a wall.
//
//
// @return: uint8_t result.
//
//****************************************************************************
extern uint8_t getCliffLeft(void);
//****************************************************************************
//
// @brief:  This command will return an integer (1 or 0) that indicates whether the
//          sensor on the left of the Roomba detects a cliff.
//
//
// @return: uint8_t result.
//
//****************************************************************************
extern uint8_t getCliffFrontLeft(void);
//****************************************************************************
//
// @brief:  This command will return an integer (1 or 0) that indicates whether the
//          sensor on the front left of the Roomba detects a cliff.
```

```
//
//
// @return: uint8_t result.
//
//*****************************************************************************
extern uint8_t getCliffFrontRight(void);
//*****************************************************************************
//
// @brief:  This command will return an integer (1 or 0) that indicates whether the
//          sensor on the front right of the Roomba detects a cliff.
//
//
// @return: uint8_t result.
//
//*****************************************************************************
extern uint8_t getCliffRight(void);
//*****************************************************************************
//
// @brief:  This command will return an integer (1 or 0) that indicates whether the
//          sensor on the right of the Roomba detects a cliff.
//
//
// @return: uint8_t result.
//
//*****************************************************************************
extern uint8_t getVirtualWall(void);
//*****************************************************************************
//
// @brief:  This command will return an integer (1 or 0) that indicates whether the
//          Roomba detects a virtual wall.
//
//
// @return: uint8_t result.
//
//*****************************************************************************
extern uint8_t getLeftWheelOvercurrent(void);
//*****************************************************************************
//
// @brief:  This command will return an integer (1 or 0) that indicates the value of
//          left wheel overcurrent sensor.
//
//
// @return: uint8_t result.
//
//*****************************************************************************
extern uint8_t getRightWheelOvercurrent(void);
//*****************************************************************************
//
// @brief:  This command will return an integer (1 or 0) that indicates the value of
//          right wheel overcurrent sensor.
//
//
// @return: uint8_t result.
//
//*****************************************************************************
extern uint8_t getMainBrushOvercurrent(void);
```

```
//*****************************************************************************
//
// @brief:  This command will return an integer (1 or 0) that indicates the value of
//          main brush overcurrent sensor.
//
//
// @return: uint8_t result.
//
//*****************************************************************************
extern uint8_t getSideBrushOvercurrent(void);
//*****************************************************************************
//
// @brief:  This command will return an integer (1 or 0) that indicates the value of
//          side brush overcurrent sensor.
//
//
// @return: uint8_t result.
//
//*****************************************************************************
extern uint8_t getDirtLevel(void);
//*****************************************************************************
//
// @brief:  This command will return an integer between 0-100 that indicates dirt
//          detection as a percentage.
//
//
// @return: uint8_t result.
//
//*****************************************************************************
extern uint8_t getInfraredCharacterOmni(void);
//*****************************************************************************
//
// @brief:  This value identifies the 8-bit IR character currently being received by
//          Roomba's omnidirectional receiver. A value of 0 indicates that no character
//          is being received.
//
//
// @return: uint8_t result.
//
//*****************************************************************************
extern uint8_t getInfraredCharacterLeft(void);
//*****************************************************************************
//
// @brief:  This value identifies the 8-bit IR character currently being received by
//          Roomba's left receiver. A value of 0 indicates that no character is being
//          received.
//
//
// @return: uint8_t result.
//
//*****************************************************************************
extern uint8_t getInfraredCharacterRight(void);
//*****************************************************************************
//
// @brief:  This value identifies the 8-bit IR character currently being received by
//          Roomba's right receiver. A value of 0 indicates that no character is being
```

```
//         received.
//
//
// @return: uint8_t result.
//
//*****************************************************************************
extern uint8_t getButtonClean(void);
//*****************************************************************************
//
// @brief:  This function returns a single bit that indicates if the clean button is
//         pressed (1 if pressed, 0 if unpressed).
//
//
// @return: uint8_t result.
//
//*****************************************************************************
extern uint8_t getButtonSpot(void);
//*****************************************************************************
//
// @brief:  This function returns a single bit that indicates if the spot button is
//         pressed (1 if pressed, 0 if unpressed).
//
//
// @return: uint8_t result.
//
//*****************************************************************************
extern uint8_t getButtonDock(void);
//*****************************************************************************
//
// @brief:  This function returns a single bit that indicates if the dock button is
//         pressed (1 if pressed, 0 if unpressed).
//
//
// @return: uint8_t result.
//
//*****************************************************************************
extern uint8_t getButtonMinute(void);
//*****************************************************************************
//
// @brief:  This function returns a single bit that indicates if the minute button is
//         pressed (1 if pressed, 0 if unpressed).
//
//
// @return: uint8_t result.
//
//*****************************************************************************
extern uint8_t getButtonHour(void);
//*****************************************************************************
//
// @brief:  This function returns a single bit that indicates if the hour button is
//         pressed (1 if pressed, 0 if unpressed).
//
//
// @return: uint8_t result.
//
//*****************************************************************************
```

```
extern uint8_t getButtonDay(void);
//****************************************************************************
//
// @brief:  This function returns a single bit that indicates if the day button is
//          pressed (1 if pressed, 0 if unpressed).
//
//
// @return: uint8_t result.
//
//****************************************************************************
extern uint8_t getButtonSchedule(void);
//****************************************************************************
//
// @brief:  This function returns a single bit that indicates if the schedule button is
//          pressed (1 if pressed, 0 if unpressed).
//
//
// @return: uint8_t result.
//
//****************************************************************************
extern uint8_t getButtonClock(void);
//****************************************************************************
//
// @brief:  This function returns a single bit that indicates if the clock button is
//          pressed (1 if pressed, 0 if unpressed).
//
//
// @return: uint8_t result.
//
//****************************************************************************
extern uint8_t getButton(BUTTON button);
//****************************************************************************
//
// @brief:  This function returns a single bit that indicates if the clean button is
//          pressed (1 if pressed, 0 if unpressed).
//   Bit  |   7  |  6   |  5  |   4  |   3  |   2  |   1  |   0
//   ------------------------------------------------------------------------
//   Value | Clock  |Schedule|  Day  |  Hour  | Minute |  Dock  |  Spot  |  Clean
//
// @return: uint8_t result.
//
//****************************************************************************
extern int16_t getDistance(void);
//****************************************************************************
//
// @brief:  The distance that Roomba has traveled in millimeters since the distance it
//          was last requested is sent as a signed 16-bit value, high byte first. This
//          is the same as the sum of the distance traveled by both wheels divided by
//          two. Positive values indicate travel in the forward direction; negative
//          values indicate travel in the reverse direction.
//
//
// @return: int16_t result.
//
//****************************************************************************
extern int16_t getAngle(void);
```

```
//*****************************************************************************
//
// @brief:  The angle in degrees that Roomba has turned since the angle was last
//          requested is sent as a signed 16-bit value, high byte first.
//          Counter-clockwise angles are positive and clockwise angles are negative.
//
//
// @return: int16_t result.
//
//*****************************************************************************
extern uint8_t getChargingState(void);
//*****************************************************************************
//
// @brief:  This function determines the current charging state of the Roomba battery.
//          The possible values returned are as follows:
//
//          0: Not charging
//          1: Reconditioning charging
//          2: Full charging
//          3: Trickle charging
//          4: Waiting
//          5: Charging fault condition
//
//
// @return: uint8_t result.
//
//*****************************************************************************
extern uint16_t getVoltage(void);
//*****************************************************************************
//
// @brief:  This function returns the voltage of the Roomba's battery in millivolts.
//
//
// @return: uint16_t result.
//
//*****************************************************************************
extern int16_t getCurrent(void);
//*****************************************************************************
//
// @brief:  This function returns the current flowing in or out of the Roomba's
//          battery in milliamps. Negative current indicates current is flowing out
//          of the battery, and positive current indicates current is flowing in.
//
//
// @return: int16_t result.
//
//*****************************************************************************
extern int16_t getTemperature(void);
//*****************************************************************************
//
// @brief:  This function returns the temperature of the Rooma's battery in degrees
//          Celsius.
//
//
// @return: int16_t result.
//
```

```
//*****************************************************************************
extern uint16_t getBatteryCharge(void);
//*****************************************************************************
//
// @brief:  This function returns the charge of the Roomba's battery in milliamp-hours.
//
//
// @return: uint16_t result.
//
//*****************************************************************************
extern uint16_t getBatteryCharge(void);
//*****************************************************************************
//
// @brief:  This function returns the estimated capacity of the Roomba's battery in
//          milliamp-hours.
//
//
// @return: uint16_t result.
//
//*****************************************************************************
extern uint16_t getWallSignal(void);
//*****************************************************************************
//
// @brief:  This function returns the strength of the wall signal.
//          Valid values are between 0-1023 inclusive.
//
//
// @return: uint16_t result.
//
//*****************************************************************************
extern uint16_t getCliffLeftSignal(void);
//*****************************************************************************
//
// @brief:  This function returns the strength of the left cliff sensor.
//          Valid values are between 0-4095 inclusive.
//
//
// @return: uint16_t result.
//
//*****************************************************************************
extern uint16_t getCliffFrontLeftSignal(void);
//*****************************************************************************
//
// @brief:  This function returns the strength of the front left cliff sensor.
//          Valid values are between 0-4095 inclusive.
//
//
// @return: uint16_t result.
//
//*****************************************************************************
extern uint16_t getCliffFrontRightSignal(void);
//*****************************************************************************
//
// @brief:  This function returns the strength of the front right cliff sensor.
//          Valid values are between 0-4095 inclusive.
//
```

```
//
// @return: uint16_t result.
//
//*****************************************************************************
extern uint16_t getCliffRightSignal(void);
//*****************************************************************************
//
// @brief:  This function returns the strength of the right cliff sensor.
//          Valid values are between 0-4095 inclusive.
//
//
// @return: uint16_t result.
//
//*****************************************************************************
extern uint8_t getChargingSourcesAvailable(void);
//*****************************************************************************
//
// @brief:  This function returns the charging sources available to the Roomba.
//          Valid values are between 0-3 inclusive.
//          0 indicates that no charging sources are available.
//          1 indicates that the internal charger is available.
//          2 indicates that the home base is available.
//          3 indicates that both the internal charger and home base are available.
//
//
// @return: uint16_t result.
//
//*****************************************************************************
extern uint8_t getOIMode(void);
//*****************************************************************************
//
// @brief:  This function returns the Roomba's current OI mode.
//          Valid values are between 0-3 inclusive.
//          0 indicates that the Roomba is off.
//          1 indicates that the Roomba is in passive mode.
//          2 indicates that the home base is available.
//          3 indicates that both the internal charger and home base are available.
//
//
// @return: uint16_t result.
//
//*****************************************************************************
extern uint8_t getSongNumber(void);
//*****************************************************************************
//
// @brief:  This function returns the currently selected OI song.
//          Valid values are between 0-15.
//
//
// @return: uint8_t result.
//
//*****************************************************************************
extern uint8_t getSongPlaying(void);
//*****************************************************************************
//
// @brief:  This function returns the state of the OI song player. 1 indicates that
```

```
//          a song is playing, 0 indicates that no song is playing.
//
//
// @return: uint8_t result.
//
//*****************************************************************************
extern uint8_t getNumberStreamPackets(void);
//*****************************************************************************
//
// @brief:  This function returns the number of data stram packets.
//
//
// @return: uint8_t result.
//
//*****************************************************************************
extern int16_t getRequestedVelocity(void);
//*****************************************************************************
//
// @brief:  This function returns the most recent radius requested with the drive
//          command in units of mm/s.
//          Valid values are between -500 to 500 inclusive.
//
//
// @return: int16_t result.
//
//*****************************************************************************
extern int16_t getRequestedRadius(void);
//*****************************************************************************
//
// @brief:  This function returns the most recent velocity requested with the drive
//          command in units of mm.
//          Valid values are between -32768 to 32767 inclusive.
//
//
// @return: int16_t result.
//
//*****************************************************************************

#ifdef __cplusplus
}
#endif

#endif /* CREATE2_UART_HEADER_H_ */
```

APPENDIX B

SOURCE CODE LISTING

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

int16_t commandBuffer[100];
int16_t responseBuffer[100];
uint8_t rxBuffer[1024];
uint16_t roombaBufferIndex = 0;
uint8_t scheduleArray[] = {CREATE2_OPCODE_SCHEDULE, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0};
uint8_t eusciFlag;
volatile bool rxFlag = 0;

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));
}

uint8_t getWheelDropLeft (void) {
   uint8_t result;
   commandBuffer[0] = CREATE2_OPCODE_SENSORS;
   commandBuffer[1] = CREATE2_PACKET_BUMP_WHEEL_DROPS;
   sendUART();
   while(!rxFlag);
   if (rxBuffer[0] & 8) {
      result = 1;
   }
   else {
      result = 0;
   }
   rxFlag = 0;
   memset(commandBuffer, 0, sizeof(commandBuffer));
   return result;
}

uint8_t getWheelDropRight (void) {
   uint8_t result;
   commandBuffer[0] = CREATE2_OPCODE_SENSORS;
```

```
    commandBuffer[1] = CREATE2_PACKET_BUMP_WHEEL_DROPS;
    sendUART();
    while(!rxFlag);
    if (rxBuffer[0] & 4) {
        result = 1;
    }
    else {
        result = 0;
    }
    rxFlag = 0;
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return result;
}

uint8_t getBumpLeft (void) {
    uint8_t result;
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_BUMP_WHEEL_DROPS;
    sendUART();
    while(!rxFlag);
    if (rxBuffer[0] & 2) {
        result = 1;
    }
    else {
        result = 0;
    }
    rxFlag = 0;
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return result;
}

uint8_t getBumpRight (void) {
    uint8_t result;
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_BUMP_WHEEL_DROPS;
    sendUART();
    while(!rxFlag);
    if (rxBuffer[0] & 1) {
        result = 1;
    }
    else {
        result = 0;
    }
    rxFlag = 0;
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return result;
}

uint8_t getWall (void) {
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_WALL;
```

```
    sendUART();
    while(!rxFlag);
    rxFlag = 0;
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return rxBuffer[0];
}

uint8_t getCliffLeft (void) {
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_CLIFF_LEFT;
    sendUART();
    while(!rxFlag);
    rxFlag = 0;
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return rxBuffer[0];
}

uint8_t getCliffFrontLeft (void) {
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_CLIFF_FRONT_LEFT;
    sendUART();
    while(!rxFlag);
    rxFlag = 0;
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return rxBuffer[0];
}

uint8_t getCliffFrontRight (void) {
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_CLIFF_FRONT_RIGHT;
    sendUART();
    while(!rxFlag);
    rxFlag = 0;
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return rxBuffer[0];
}

uint8_t getCliffRight (void) {
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_CLIFF_RIGHT;
    sendUART();
    while(!rxFlag);
    rxFlag = 0;
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return rxBuffer[0];
}

uint8_t getVirtualWall (void) {
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_VIRTUAL_WALL;
    sendUART();
```

```
    while(!rxFlag);
    rxFlag = 0;
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return rxBuffer[0];
}

uint8_t getLeftWheelOvercurrent (void) {
    uint8_t result;
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_WHEEL_OVERCURRENTS;
    sendUART();
    while(!rxFlag);
    rxFlag = 0;
    if (rxBuffer[0] & 16) {
        result = 1;
    }
    else {
        result = 0;
    }
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return result;
}

uint8_t getRightWheelOvercurrent (void) {
    uint8_t result;
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_WHEEL_OVERCURRENTS;
    sendUART();
    while(!rxFlag);
    rxFlag = 0;
    if (rxBuffer[0] & 8) {
        result = 1;
    }
    else {
        result = 0;
    }
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return result;
}

uint8_t getMainBrushOvercurrent (void) {
    uint8_t result;
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_WHEEL_OVERCURRENTS;
    sendUART();
    while(!rxFlag);
    rxFlag = 0;
    if (rxBuffer[0] & 4) {
        result = 1;
    }
    else {
```

```
        result = 0;
    }
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return result;
}

uint8_t getSideBrushOvercurrent (void) {
    uint8_t result;
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_WHEEL_OVERCURRENTS;
    sendUART();
    while(!rxFlag);
    rxFlag = 0;
    if (rxBuffer[0] & 1) {
        result = 1;
    }
    else {
        result = 0;
    }
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return result;
}

uint8_t getDirtLevel (void) {
    uint8_t result;
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_DIRT_DETECT;
    sendUART();
    while(!rxFlag);
    rxFlag = 0;
    memset(commandBuffer, 0, sizeof(commandBuffer));
    result = (rxBuffer[0]/255) * 100;
    return result;
}

uint8_t getInfraredCharacterOmni (void) {
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_INFRARED_CHARACTER_OMNI;
    sendUART();
    while(!rxFlag);
    rxFlag = 0;
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return rxBuffer[0];
}

uint8_t getInfraredCharacterLeft (void) {
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_INFRARED_CHARACTER_LEFT;
    sendUART();
    while(!rxFlag);
    rxFlag = 0;
```

```
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return rxBuffer[0];
}

uint8_t getInfraredCharacterRight (void) {
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_INFRARED_CHARACTER_RIGHT;
    sendUART();
    while(!rxFlag);
    rxFlag = 0;
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return rxBuffer[0];
}

uint8_t getButton (BUTTON button) {
    uint8_t result;
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_BUTTONS;
    sendUART();
    while(!rxFlag);
    rxFlag = 0;
    if (button != 255) {
        if (rxBuffer[0] & button) {
            result = 1;
        }
        else {
            result = 0;
        }
    }
    else {
        result = rxBuffer[0];
    }
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return result;
}

int16_t getDistance (void) {
    int16_t result;
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_DISTANCE;
    sendUART();
    while(!rxFlag);
    rxFlag = 0;
    result = (rxBuffer[0] << 8) | (rxBuffer[1]);
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return result;
}

int16_t getAngle (void) {
    int16_t result;
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
```

```
    commandBuffer[1] = CREATE2_PACKET_ANGLE;
    sendUART();
    while(!rxFlag);
    rxFlag = 0;
    memset(commandBuffer, 0, sizeof(commandBuffer));
    result = (rxBuffer[0] << 8) | (rxBuffer[1]);
    return result;
}

uint8_t getChargingState (void) {
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_CHARGING_STATE;
    sendUART();
    while(!rxFlag);
    rxFlag = 0;
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return rxBuffer[0];
}

uint16_t getVoltage (void) {
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_VOLTAGE;
    sendUART();
    while(!rxFlag);
    rxFlag = 0;
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return (rxBuffer[0] << 8) | (rxBuffer[1]);
}

int16_t getCurrent (void) {
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_CURRENT;
    sendUART();
    while(!rxFlag);
    rxFlag = 0;
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return (rxBuffer[0] << 8) | (rxBuffer[1]);
}

int16_t getTemperature (void) {
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_TEMPERATURE;
    sendUART();
    while(!rxFlag);
    rxFlag = 0;
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return rxBuffer[0];
}

uint16_t getBatteryCharge (void) {
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
```

```
    commandBuffer[1] = CREATE2_PACKET_TEMPERATURE;
    sendUART();
    while(!rxFlag);
    rxFlag = 0;
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return (rxBuffer[0] << 8) | (rxBuffer[1]);
}

uint16_t getBatteryCapacity (void) {
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_BATTERY_CAPACITY;
    sendUART();
    while(!rxFlag);
    rxFlag = 0;
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return (rxBuffer[0] << 8) | (rxBuffer[1]);
}

uint16_t getWallSignal (void) {
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_WALL_SIGNAL;
    sendUART();
    while(!rxFlag);
    rxFlag = 0;
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return (rxBuffer[0] << 8) | (rxBuffer[1]);
}

uint16_t getCliffLeftSignal (void) {
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_CLIFF_LEFT_SIGNAL;
    sendUART();
    while(!rxFlag);
    rxFlag = 0;
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return (rxBuffer[0] << 8) | (rxBuffer[1]);
}

uint16_t getCliffFrontLeftSignal (void) {
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_CLIFF_FRONT_LEFT_SIGNAL;
    sendUART();
    while(!rxFlag);
    rxFlag = 0;
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return (rxBuffer[0] << 8) | (rxBuffer[1]);
}

uint16_t getCliffFrontRightSignal (void) {
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_CLIFF_FRONT_RIGHT_SIGNAL;
```

```
    sendUART();
    while(!rxFlag);
    rxFlag = 0;
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return (rxBuffer[0] << 8) | (rxBuffer[1]);
}

uint16_t getCliffRightSignal (void) {
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_CLIFF_RIGHT_SIGNAL;
    sendUART();
    while(!rxFlag);
    rxFlag = 0;
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return (rxBuffer[0] << 8) | (rxBuffer[1]);
}

uint8_t getChargingSourcesAvailable (void) {
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_CHARGING_SOURCES_AVAILABLE;
    sendUART();
    while(!rxFlag);
    rxFlag = 0;
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return rxBuffer[0];
}

uint8_t getOIMode(void) {
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_CHARGING_SOURCES_AVAILABLE;
    sendUART();
    while(!rxFlag);
    rxFlag = 0;
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return rxBuffer[0];
}

uint8_t getSongNumber(void) {
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_SONG_NUMBER;
    sendUART();
    while(!rxFlag);
    rxFlag = 0;
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return rxBuffer[0];
}

uint8_t getSongPlaying(void) {
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_SONG_PLAYING;
    sendUART();
```

```
    while(!rxFlag);
    rxFlag = 0;
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return rxBuffer[0];
}

uint8_t getNumberStreamPackets(void) {
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_NUMBER_STREAM_PACKETS;
    sendUART();
    while(!rxFlag);
    rxFlag = 0;
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return rxBuffer[0];
}

int16_t getRequestedVelocity(void) {
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_REQUESTED_VELOCITY;
    sendUART();
    while(!rxFlag);
    rxFlag = 0;
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return (rxBuffer[0] << 8) | (rxBuffer[1]);
}

int16_t getRequestedRadius(void) {
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_REQUESTED_RADIUS;
    sendUART();
    while(!rxFlag);
    rxFlag = 0;
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return (rxBuffer[0] << 8) | (rxBuffer[1]);
}

int16_t getRequestedRightVelocity(void){
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_REQUESTED_RIGHT_VELOCITY;
    sendUART();
    while(!rxFlag);
    rxFlag = 0;
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return (rxBuffer[0] << 8) | (rxBuffer[1]);
}

int16_t getRequestedLeftVelocity(void) {
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_REQUESTED_LEFT_VELOCITY;
    sendUART();
    while(!rxFlag);
```

```
    rxFlag = 0;
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return (rxBuffer[0] << 8) | (rxBuffer[1]);
}

uint16_t getLeftEncoderCounts(void) {
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_LEFT_ENCODER_COUNTS;
    sendUART();
    while(!rxFlag);
    rxFlag = 0;
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return (rxBuffer[0] << 8) | (rxBuffer[1]);
}

uint16_t getRightEncoderCounts(void) {
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_RIGHT_ENCODER_COUNTS;
    sendUART();
    while(!rxFlag);
    rxFlag = 0;
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return (rxBuffer[0] << 8) | (rxBuffer[1]);
}

uint8_t getLightBumper (LIGHTBUMPER lightBumper) {
    uint8_t result;
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_BUTTONS;
    sendUART();
    while(!rxFlag);
    rxFlag = 0;
    if (lightBumper != 63) {
        if (rxBuffer[0] & lightBumper) {
            result = 1;
        }
        else {
            result = 0;
        }
    }
    else {
        result = rxBuffer[0];
    }
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return result;
}

uint16_t getLightBumpLeftSignal(void) {
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_LIGHT_BUMP_LEFT_SIGNAL;
    sendUART();
```

```
    while(!rxFlag);
    rxFlag = 0;
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return (rxBuffer[0] << 8) | (rxBuffer[1]);
}

uint16_t getLightBumpFrontLeftSignal(void) {
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_LIGHT_BUMP_FRONT_LEFT_SIGNAL;
    sendUART();
    while(!rxFlag);
    rxFlag = 0;
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return (rxBuffer[0] << 8) | (rxBuffer[1]);
}

uint16_t getLightBumpCenterLeftSignal(void) {
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_LIGHT_BUMP_CENTER_LEFT_SIGNAL;
    sendUART();
    while(!rxFlag);
    rxFlag = 0;
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return (rxBuffer[0] << 8) | (rxBuffer[1]);
}

uint16_t getLightBumpCenterRightSignal(void) {
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_LIGHT_BUMP_CENTER_RIGHT_SIGNAL;
    sendUART();
    while(!rxFlag);
    rxFlag = 0;
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return (rxBuffer[0] << 8) | (rxBuffer[1]);
}

uint16_t getLightBumpFrontRightSignal(void) {
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_LIGHT_BUMP_FRONT_RIGHT_SIGNAL;
    sendUART();
    while(!rxFlag);
    rxFlag = 0;
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return (rxBuffer[0] << 8) | (rxBuffer[1]);
}

uint16_t getLightBumpRightSignal(void) {
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_LIGHT_BUMP_RIGHT_SIGNAL;
    sendUART();
    while(!rxFlag);
```

```
    rxFlag = 0;
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return (rxBuffer[0] << 8) | (rxBuffer[1]);
}

int16_t getLeftMotorCurrent(void) {
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_LEFT_MOTOR_CURRENT;
    sendUART();
    while(!rxFlag);
    rxFlag = 0;
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return (rxBuffer[0] << 8) | (rxBuffer[1]);
}

int16_t getRightMotorCurrent(void) {
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_RIGHT_MOTOR_CURRENT;
    sendUART();
    while(!rxFlag);
    rxFlag = 0;
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return (rxBuffer[0] << 8) | (rxBuffer[1]);
}

int16_t getMainBrushMotorCurrent(void) {
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_MAIN_BRUSH_MOTOR_CURRENT;
    sendUART();
    while(!rxFlag);
    rxFlag = 0;
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return (rxBuffer[0] << 8) | (rxBuffer[1]);
}

int16_t getSideBrushMotorCurrent(void) {
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_SIDE_BRUSH_MOTOR_CURRENT;
    sendUART();
    while(!rxFlag);
    rxFlag = 0;
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return (rxBuffer[0] << 8) | (rxBuffer[1]);
}

uint8_t getStasis(void) {
    commandBuffer[0] = CREATE2_OPCODE_SENSORS;
    commandBuffer[1] = CREATE2_PACKET_STASIS;
    sendUART();
    while(!rxFlag);
    rxFlag = 0;
```

```
    memset(commandBuffer, 0, sizeof(commandBuffer));
    return rxBuffer[0];
}
// UART interrupt service routine
void EUSCIA2_IRQHandler(void)
{
    int i;
    rxFlag = 1;
    if (EUSCI_A2->IFG & EUSCI_A_IFG_RXIFG)
    {
        for (i = 0; i < 1024; i++){
            rxBuffer[i] = 0;
        }
        rxBuffer[roombaBufferIndex] = EUSCI_A2->RXBUF;
        roombaBufferIndex++;
        // Check if the TX buffer is empty first
        //while(!(EUSCI_A2->IFG & EUSCI_A_IFG_TXIFG));
    }
    EUSCI_A2->IFG &= ~EUSCI_A_IFG_RXIFG;
}
```

REFERENCES

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

Zachary Holloway is a senior in electrical engineering at the University of Texas at Arlington. In addition to his progress on his B.S. in electrical engineering, he also has a minor in math and is currently taking graduate courses in electrical engineering through the fast-track program. This driver is the second one he has written; the first being a driver interfacing the Pixy2 camera and the MSP432P401R microcontroller. He is currently working on a senior design project called the Sensorium, a project sponsored by Raytheon, which is a super-sensor device with over 15 onboard sensors and a software-defined radio. He has also worked on a number of research projects in the past, including helping a biomedical engineering group at UTA create a wearable magnetic sensor with tactile feedback, and a junior design project that was a heart rate alarm clock. He currently works as a teaching assistant for the department of electrical engineering at UTA teaching advanced microcontrollers, and has previously worked for UTA in the Department of Physics designing LabVIEW programs to test low-voltage power supplies destined for the ATLAS Tile Calorimeter in the Large Hadron Collider. He is currently the President of UTA's Eta Kappa Nu chapter and Vice President of UTA's Tau Beta Pi chapter.

He plans to pursue an M.S. in electrical engineering immediately after graduating, and possible a Ph.D. at some point in the future. He has a strong interest in embedded systems and would like to make it his primary field of study. Once he moves to the industry for a career, he would like to be an applications engineer in the field of embedded systems.