Remote Car

Project Number: 8

Dr.Schmalzel

Marc Giordano, Nathan Sulzer, Andrei Creanga

12/20/17

Introduction:

Over the course of this class we have become familiar with programing the outputs/inputs of our MSP430 processors, working with serial communication, and interacting with signals from the outside world with ADCs. Now we are at the point where we can put them together to create functioning open and closed loop systems that enable us to create functional electronic devices ready for the consumer market. Our project consists of the base for what will eventually become a butter passing robot based on a hit television show. Currently our system is open loop allowing UART serial communication to input commands to the car along with using ADCs that read from an analog joystick. Using both of these functions we can allow for control of the car by a user input and eventually an additional circuit that we intend to reuse from the original tank.

Design Specification:

The UART currently operates by using the following commands to change the direction of the car currently these are just placeholders to test out that our code can functionally change the cars direction using commands input from UART:

0x00000= stop command
0x00001= forward command
0x00002= reverse movement
0x00003= left command (right wheel goes forward and left wheel goes in reverse)
0x00004= right command (right wheel goes in reverse left wheel goes forward)
0x00005= take input from joystick
0x00006= take input from bluetooth app "punchthrough"

If all goes according to plan, then the car should move in the direction specified by the value input by the user over UART. The car will continue to move in that direction until a new direction is inputted, or it is told to stop.

The analog and digital commands being used to control the car had a conflict in control with our original design so we needed to add the last 2 functions in order to allow for us to switch which peripheral was in control at a given time this made it possible for our analog joystick to control it at times and our UART commands to control it when the UART was turned off.

Design Approach:

The basis of the hardware used, in which the rest of the hardware sat upon, was the car frame. This contained four wheels, two of which had motors attached, that were wired up to the L293D motor driver, and by extension, the MSP430FR6989 microprocessor. Between the MSP430FR6989 and the L293D was a Fuel Tank MKII Battery BoosterPack Plug-In Module by TI, to power the car. Also included was a RQS25 joystick used to control direction of the motor driver as well as voltage applied to it.

A majority of the code used was based directly on our Milestone 2 code which is also why we choose the FR6989 due to familiarity with it and the prior examples we could look back on for reference. In the case of a UART connection only one movement command can be implemented over UART at a time (Forward, Reverse, Rotate Left, Rotate Right). This actually gets implemented using hexadecimal which is the easiest way of sending numbers to the device over UART. Reverse is implemented using a H Bridge (a 2 bit logic based switch) which will invert the polarity of the high and low input signal and allow the bots motors to move in reverse. The H bridge was connected to each motor driver independently, allowing one wheel to rotate forward while the other rotates back. Rotate Left and Right is done by using an H bridge to only reverse 1 motor driver direction at a time. The code needed for rotate left and right were added after the initial functions of forward and reverse movement were added and working. Rotating functioned by the H bridge reversing the direction of one motor at a time, when each motor moved the result was the bot rotating in the direction of which motor was moving in reverse.

For the original design of the bot that would be implemented by using the same basic implementation for our motor driver but the IC that was to be used was the

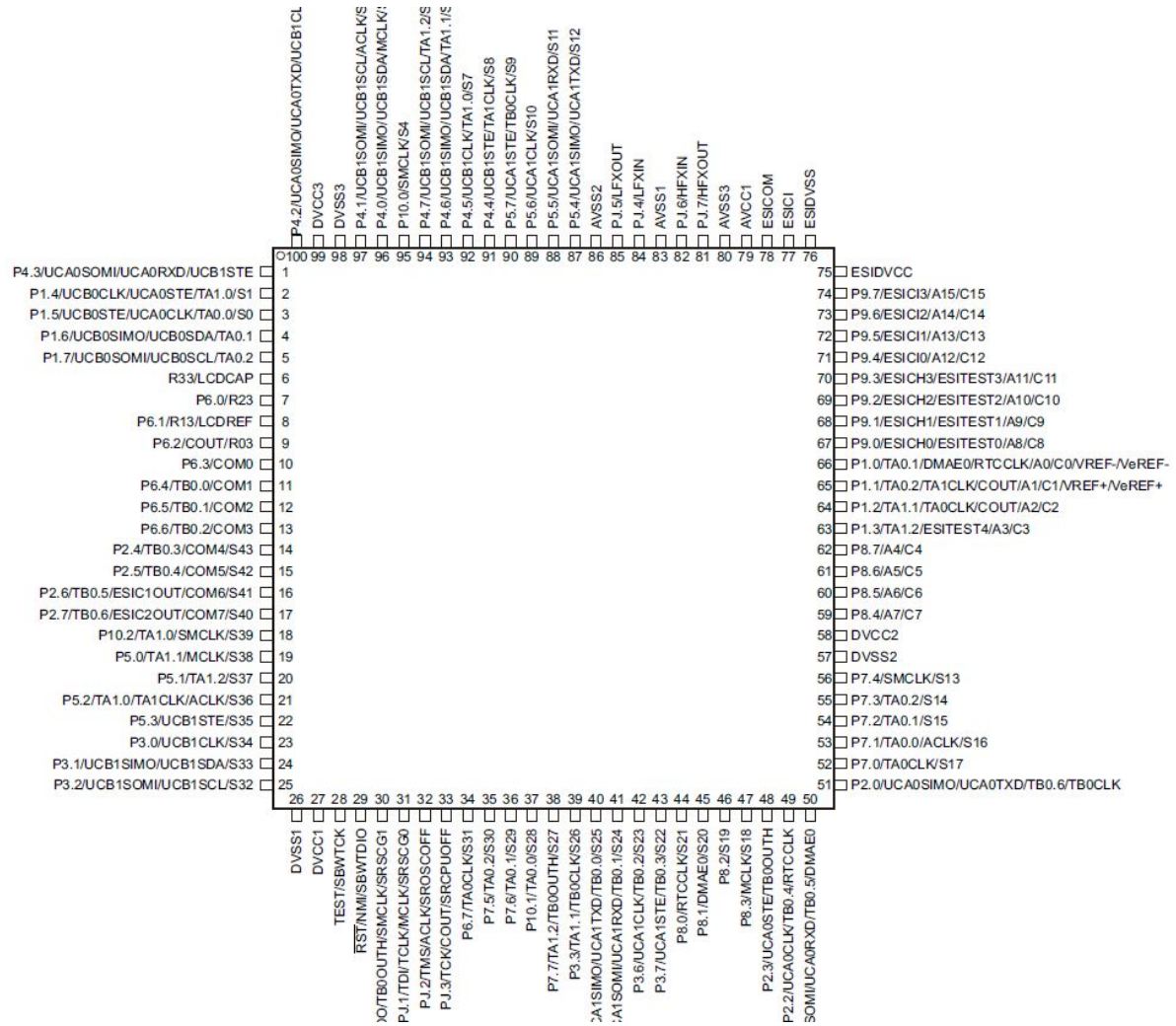

Figure 1: MSP430FR6989 PIN assignments

Figure 1 above shows the pinouts and the registers they're connected to in the MSP430FR6989 databoard.

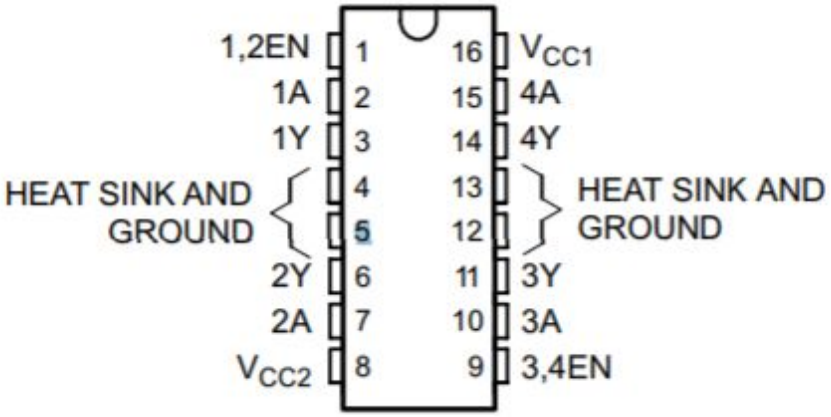

Figure 2: Pin layout for L293D Motor Driver

Figure 2 shows the pin layout of the motor driver uses to drive each motor as well as switch directions of operation of each motor.

UART was the original control mechanism for the RC bot. Other functionality was added after UART was able to work. As shown in the design specifications each input over UART correlates to a movement of the RC bot. The way the bot would actually decipher each UART command to a movement is by first taking our designated output pins (9 and 4) of the 6989 and relating them to Bit4 and Bit7. Outpin pins 9 and 4 were the hardware pins on the 6989 and Bit 4 and Bit 7 respectively correlates to which bit in the 12 ADC line the bit is which then correlates to the output of the left and right wheel respectively.
With each value of the bit over UART the responding case statement would indicate the movement.
If UART input = 0x00000 This indicates a stop
The case statement in software will then equal:

```
case 0x0000 : //stop
    {
        P9OUT &= ~BIT4;
        P4OUT &= ~BIT7;
        break;
    }
```

Figure 3: Case statement for stopping

This indicates that P9 and P4 will return a 0, which will turn the PWM signal to each motor to 0 which will stop the RC car.
If UART = 0x00001

This indicates that P9 and P4 correlate to BIT4 and BIT7 respectively.
P2OUT and P3OUT indicate the direction of each motor. P2OUT correlates to the left wheel, P3OUT correlates to the right wheel. The exact value returned for P2OUT and P3OUT is also important. In this case of UART input P2out = 0X10 indicates the left wheel will turn forward, and P3OUT= 0x01 indicates the right wheel will turn forward.

For each case of UART from 0x0001-0x0004 the direction of the left and right wheel will change. This is achieved by connecting the msp430 through an L293D H bridge chip. This chip is 2 bits with each bit corresponding to 1 motor driver. The input to output relations of the 2 bit L293D chip is shown below.

00= forward
11= reverse
01= turn left
10= turn right
The way these movements are achieved will be explained below with each UART statement.

```
case 0x0001 : //Foward
    {
        P9OUT |= BIT4;
        P4OUT |= BIT7;
        P2OUT = 0x10;
        P3OUT = 0x01;
    break;
    }
```

Figure 4: Case statement for forward

This code causes the bot to move forward as a result of the L293D chip being left to 00, meaning the original direction of the motors is left. While both motor drivers are set to a 5.5V allowing them to turn.

UART 0x0002 = backward

```
case 0x0002 : //Backwards
    {
        P9OUT |= BIT4;
        P4OUT |= BIT7;
        P2OUT = 0x08;
        P3OUT = 0x02;
    break;
    }
```

Figure 5: Case statement for backwards

P2OUT= 0x08 means the left wheel will turn in reverse and P3OUT=0X02 means the right wheel will turn in reverse. This is a result of the L293D chip being set to “11” which will flip the direction that each motor will rotate. The 5V from the msp430 will then run to each motor driver allowing the bot to move in reverse.

UART = 0x0003 turning right.

```
case 0x0003 : //Right
    {
        P9OUT |= BIT4;
        P4OUT |= BIT7;
        P2OUT = 0x10;
        P3OUT = 0x02;
        break;
    }
```

Figure 6: Case statement for turning right

When P2OUT= 0x10 that indicates the left wheel is turning forward, as originally set. However the H bridge chip is set to “01”, only switching the direction of the right wheel. The end result is the left wheel moves forward while P3OUT=0x02 moves the right wheel in reverse. This output correlates to the RC bot rotating clockwise.

UART=0x004 turning left

```
case 0x0004 : //Left
    {
        P9OUT |= BIT4;
        P4OUT |= BIT7;
        P2OUT = 0x08;
        P3OUT = 0x01;
        break;
    }
```

Figure 7: Case statement for turning left

The code for turning left is opposite of turning right. The L293D chip is set to “10”, reversing the direction of the left motor wheel but not the right. The value of P2OUT=0x08 indicates the motor wheel turning in reverse because P2OUT=0X10 indicates the wheel turning forward. As a result of this UART case the right wheel will turn forward (P3OUT=0X01) and the left wheel will turn in reverse. This will rotate the bot counterclockwise.

UART=0x005 and UART = 0X006

```
case 0x0005 : //UartON
     {
      UARTContrl = 0x00;
      break;
     }
case 0x0006 : //UartOff
     {
        UARTContrl = 0x01;
        break;
     }
```

Figure 8: Case statements for UART control

This code is for further implementation of methods of control. Case 0x005 turns UART on while case 0X006 turns UART off. The reason for this is because in testing with other methods of control (joystick and the lightblue app) could not take inputs from both at the same time. By turning UART off the msp430 will only take inputs from either the joystick or the bluetooth app which expands user input options.

The joystick: The joystick is a analog system of control directly wired into the msp430fr6989 board. The code below shows the relationship between x direction inputs and y direction inputs and their relation to one another. The output value of the analog joystick was converted to a hex value, which was then put in place of the UART value (which is why UART must be turned off for the joystick to function).

```
void SteeringSetAnalogXY(void)
{
    if(CurrentValueY >= 0x0F00)
    {
     P9OUT |= BIT4;
     P4OUT |= BIT7;
     P2OUT = 0x10;
     P3OUT = 0x01;

    } else if(CurrentValueY <= 0x0010)
    {
    P9OUT |= BIT4;
    P4OUT |= BIT7;
    P2OUT = 0x08;
    P3OUT = 0x02;

    } else if(CurrentValueX >= 0x0F00)
    {
        P9OUT |= BIT4;
                    P4OUT |= BIT7;
                    P2OUT = 0x10;
                    P3OUT = 0x02;

    } else if(CurrentValueX <= 0x0010)
    {
        P9OUT |= BIT4;
                    P4OUT |= BIT7;
                    P2OUT = 0x08;
                    P3OUT = 0x01;
```

Figure 9: Joystick analog controller code

The direction of the joystick will put out the UART code corresponding to each direction desired by the user. The function output remain the same in this case, the only change occurs with where the HEX code input is coming from.

Processor Selection

The processor selected for this implementation is the MSP430FR6989. This board was narrowed down by eliminating the other four boards at our disposal:
MSP430FR2311: Memory count is too low and did not have an LCD screen, as well has not having enough in/out pins.
MSP430G2553: Programming issues. Personal preference, harder to program.
MSP430FR5994: Do not need the super capacitor. Waste of resources.
MSP430F5529: Did not contain the LCD screen, or have enough in/out pins.
From these decisions, it was decided to use the MSP430FR6989, even though we never ended up using the LCD screen, we had already developed the system around the 6989.

PCB:

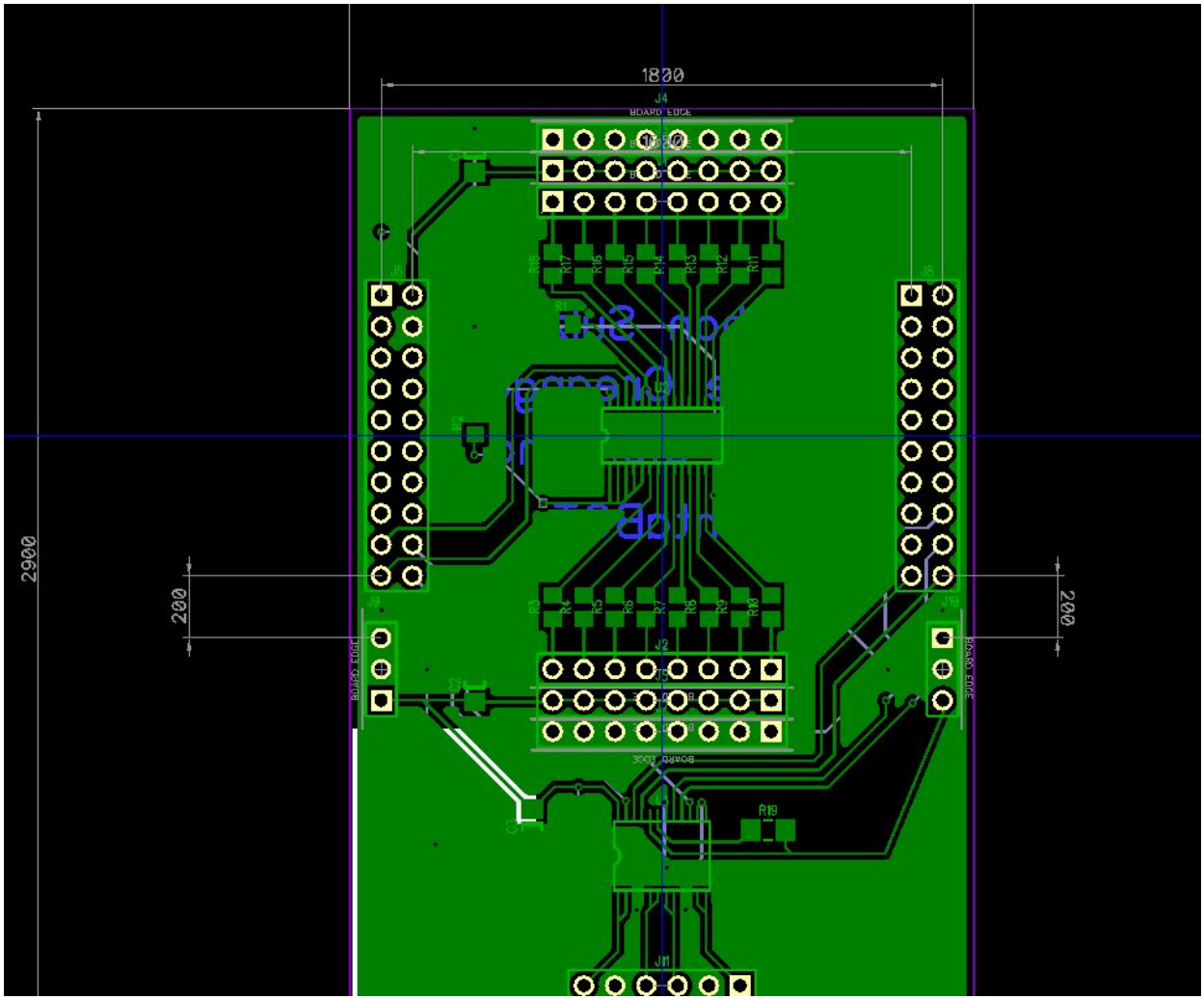

Figure 10: PCB layout

Our design originally included a PCB in order to simply the connections of the data processor to the motor drivers however due to missing capacitors that were needed as well as lack of access to the pick and place lab by the time of receiving the PCB our project required working around the absence of PCB implementation in order to function. Talking specifics of the PCB, our PCB was simply a way to connect the desired pins of the MSP430FR6989 databoard to the motor drivers and also incorporate an H bridge in order to switch direction of each motor independently. The implementation of the PCB would have vastly simplified the wiring of the databoard to battery pack to L235D chip however we had to make due without the PCB due to lacking components. As seen above the left and right side of the pcb each connected to the male output pins of the databoard. The bottom,top, and center connections on the PCB would handle connections from the databoard to the battery pack. Where the battery pack extension would finally finish the connections to the motors. Also the pins

at the bottom would handle any inputs we later created in order to send the UART signals. In our original design this included a joystick as well as a bluetooth connection. As our PCB was incomplete we did not include the PCB in our final product however the joystick and bluetooth connection were both created and working. The center chip on this PCB is the PCA 9685 chip, which is a 16 channel PWM chip. This chip was chosen in order to handle the levels of PWM input originally desired in case of the original design. The original design included different functioning speeds as well as direction. This was to be implemented similar to milestone 2 with FF equaling full speed (5V) and 00 equaling stop (0V), however without the PCB and with the mindset of the project changed in that each direction would receive the full 5V when functioning instead of different strengths. The TB6612FNG chip is implemented in the bottom of the board as a DC motor control. This was to connect the databoard to the motors. Again because of the lacking of PCB implementation the design went around this and instead used the original motor controls included with the ispy mini RC car.

Results and Conclusion:

When presenting, our presentation went off almost perfectly. We had some last second issues regarding the car working for the presentation demo, but in the end, everything worked well. The car was successfully able to perform all of the required actions including moving forward and backward, turning left and right, and stopping. This project was a lot of fun to work on, and used almost everything we learned throughout the semester.

References:

MSP430FR6989 Datasheet:
http://www.ti.com/lit/ds/symlink/msp430fr6989.pdf
L293D Motor Driver Datasheet:
http://www.ti.com/lit/ds/symlink/l293.pdf

Appendix:

C was the main and only type of coding used in this project.
Realterm was used to communicate with the MSP430FR6989 through UART.

Figure 11: Remote car operational