Goals for this Lab Assignment:

1. Work with wheel angles measured by the motor’s Optical Encoders and write to both PWM and DAC output channels.
2. Estimate Velocity from Optical Encoder Position Feedback.
3. Identify the friction in the robot’s drive system.

SYS/BIOS Objects Used:

Clock, Swi

Library Functions Used:

Global variables Enc1_rad, Enc2_rad, Enc3_rad and Enc4_rad. Functions
void writeDAC7564(float dac1, float dac2), void PWM_out(enum epwm ep, float u),
void init_PWM(enum epwm ep)

Prelab: Read the first few pages of the TMS320F28335datasheet
http://coeecs.ece.illinois.edu/ge423/datasheets/F28335Ref_Guides/F28335DataSheet.pdf to familiarize yourself with the robot’s motor control processor.

Laboratory Exercise

Up to this point, lecture and lab have been focused on the MSP430G2553 microcontroller used in your homework assignments and the dual core OMAPL138 processor on your robot vehicle. In this lab and also lab assignments 4 and 5 we are going to switch gears a bit and focus on another processor on the robot’s circuit board that takes care of the motor control and acquisition of many of the robot’s sensors. This chip is named the TMS320F28335. Texas Instruments categorizes this processor as a microcontroller but just for marketing reasons. It is actually one of TI’s first DSP series. Looking at its datasheet http://coeecs.ece.illinois.edu/ge423/datasheets/F28335Ref_Guides/F28335DataSheet.pdf and Figure 1 below it is much more powerful than a standard microcontroller. The TMS320F28335 is clocked at 150MHz, has 67Kb of internal RAM, 512Kb of program flash and a large number of peripherals. The peripherals we are most interested in for our robot vehicle are the ADC (Analog to Digital) inputs, Optical Encoder inputs, PWM outputs and SPI and I2C serial ports. In addition to the on-chip peripherals of the TMS320F28335, external chips and sensors have been added to the robot’s circuit board to interface with the TMS320F28335 giving it even more capabilities. See Table 1 for a list of inputs, outputs and sensors the TMS320F28335 provides for the robot.
3 Functional Overview

Figure 1: Pictorial of the Many Capabilities of the TMS320F28335 Processor

Figure 3-1. Functional Block Diagram
<table>
<thead>
<tr>
<th>On Chip Peripherals</th>
<th>Off Chip Peripherals</th>
<th>Sensors/Actuators Interfaced by these Peripherals</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. 12 PWM Outputs</td>
<td>1. 4 Optical Encoder Inputs</td>
<td>1. +/- 400 deg/s Rate Gyro</td>
</tr>
<tr>
<td>2. 16 ADC Inputs</td>
<td>2. 2 DAC Outputs</td>
<td>2. Ultrasonic Distance Sensors</td>
</tr>
<tr>
<td>3. 2 Optical Encoder Inputs</td>
<td></td>
<td>3. IR Distance Sensors</td>
</tr>
<tr>
<td>4. SPI serial port</td>
<td></td>
<td>4. Compass</td>
</tr>
<tr>
<td>5. I2C serial port</td>
<td></td>
<td>5. 6 Optical Encoder Sensors</td>
</tr>
<tr>
<td>6. UART serial port</td>
<td></td>
<td>6. 6 RC Servos</td>
</tr>
<tr>
<td>7. McBSP serial port</td>
<td></td>
<td>7. 2 DC Motors</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8. Text LCD Screen</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9. 4 Toggle Switches</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10. 2 LEDs</td>
</tr>
</tbody>
</table>

Table 1: TMS320F28335 Capabilities used by Mechatronics Robot Vehicle.

In this lab we are not going to focus on learning a new aspect of the SYS/BIOS real-time operating system, instead we are going to focus on using the Optical Encoder inputs, DAC (Digital to Analog Converter) outputs and PWM (Pulse Width Modulation) outputs. In Lab 4 we will take a more detailed look at the source code given by the F28335 project creator and also study the circuit board design of the robot’s processor board.

So you are not completely in the dark about the source code given, a brief description is in order. To read four of the possible six optical encoder sensors that can be connected to the robot’s processor board, four LS7366 chips are interfaced to the TMS320F28335 through its SPI serial peripheral. These LS7366 chips are 32 bit counters that count up when the optical encoder is turned in the positive direction and count down when turned in the opposite direction. So the TMS320F28335 uses its SPI serial port to read this 32 bit count from each of the LS7366 chips. The TMS320F28335 has a peripheral for generating PWM signals so no addition circuits are needed for PWM outs. In the case of the DAC outputs, the TMS320F28335 does not have a DAC peripheral. So in addition to the LS7366 chips connected to the TMS320F28335’s SPI serial port there is a DAC7564 chip connected to the SPI serial interface giving the TMS320F28335 two channels of 0-2.5V 12bit DAC output.

The TMS320F28335 code’s purpose is twofold. One is to acquire all the sensor information every 1 millisecond and communicate this information to the OMAPL138 through another serial port called the McBSP serial port. Second is to perform the speed control of the robot’s motors also every 1 millisecond. In lab 4 you will be developing code to add the acquisition of the sensors connected to the ADC peripheral of the TMS320F28335. In lab 5 you will develop the motor control algorithm.

So in the TMS320F28335 project creator’s default code, what happens each 1 millisecond?

1. A Clock object calls its function void start_dataCollection(void) every 1 millisecond. Inside this function all that is done is call the function start_SPI(). Start_SPI() selects all four of the LS7366 chips and sends a command to them over the SPI to latch the current reading of their connected optical encoder. Start_SPI() does not wait for the 8bits of data to be transmitted. It starts the SPI transfer in the SPI peripheral unit. After Start_SPI() initiates the transfer, the
function ends, leaving the processor free for other tasks while the SPI peripheral handles the SPI transfer. Once the transfer is complete, the SPI’s receive interrupt takes care of the next steps in reading the four LS7366 chips.

2. After transmitting the 8 bit command to latch the four LS7366 readings, the SPI serial port generates a Receive interrupt (we are not using the SPI Transmit interrupt with the F28335) indicating that the transfer is complete. On receiving this interrupt the HWI function void SPI_RXint(void) is called.

3. Inside the SPI_RXint interrupt function, the TMS320F28335 selects the first LS7366 chip and sends it a command to return its 32bit count value.

4. On receiving the SPI’s next interrupt the SPI_RXint interrupt function knows that the 32bit count reading is sitting in the SPI’s FIFO buffer. After reading the data from the first LS7366 chip, the function will select the second LS7366 chip and sends it the command to return its 32bit count value.

5. This reading of the remaining LS7366 chips repeats three more times to acquire the count readings of the remaining 3 LS7366 chips.

6. On the final SPI receive interrupt that indicates the fourth count value has been communicated from the fourth LS7366 chip the SWI swi_control is posted. This Swi object’s function is void control(void).

7. Here in this function void control(void) is where you will be putting your own code, but before your code the function updateData(); needs to be called. updateData() checks if there is any new sensor readings from the slow sensors like the compass and IR distance sensors and communicates those values to your code in predefined global variables.

8. After updateData() returns is where you should insert your code. At this point you have all the sensor readings at your access.

9. After your code you should call the function sendData(). This function communicates all the data sampled by the TMS320F28335 to the OMAPL138’s DSP core through the McBSP (Multi Channel Buffered Serial Port) serial port.

10. Then when 1 millisecond occurs again the entire process is repeated.

Your robot can only be sitting in two locations: on the floor or on a stand on the bench with all wheels off the bench.

Failure to do so may result in robot suicide, i.e. the robot may suddenly drive itself off the bench. If a student catastrophically breaks their robot, both the instructor’s and the student’s lab work will triple. Both will still have to come to lab, both will have to come after lab hours to use different group’s robot to complete the lab, and both will have to come in after hours to repair the robot so it can be used in other lab sections.

Exercise 1: Basic Input and Output

1. Create a new project using the Sp2019F28335ProjectCreator.exe executable found in \Spr2019F28335ProjectCreator directory of your repository.

2. As just summarized above the function void control(void) is the function for the Swi object swi_control. swi_control is posted every 1ms after the 4 LS7366 Optical Encoder interface chips have been read. You will be placing all the code you develop for this lab inside the control() function and in between the functions
updateData() and sendData(). At that point in the function control(), the motor’s optical encoder sensors have been measured and are located in the variables Enc1_rad, Enc2_rad. There are also two other encoder readings sampled, Enc3_rad and Enc4_rad which are unconnected for Exercise 1. For this short exercise think of the wheels of the robot as input dials that change a numerical output value when rotated. We will send this numerical output value to the DAC and the PWM channels to test that they are working. Add the following functionality to the control() function:

a. Normally we will be using PWM4 and PWM5 to drive RC servo motors. For laboratory 3 only we will setup PWM4 and PWM5 to produce the same 20KHz carrier frequency PWM signal the drives the two robot motors. This is done so that you can see the PWM signal on the oscilloscope. So, towards the top of your main() function find the source lines init_PWM_AS_RCServo(EPWM4B); and init_PWM_AS_RCServo(EPWM5); and change them to init_PWM(EPWM4); and init_PWM(EPWM5);

b. Every 1 ms (which is every time this function is called), send the radian value of encoder channels 1 & 2 to the PWM output channels 4 & 5. Also send a scaled value of the encoder channels 1 & 2 to the two DAC channels. In future labs you can use the DAC as a debug output to monitor the control effort sent over a PWM channel to a DC Motor’s amplifier. The range of the PWM command is between -10 and 10. Our DAC can only output a range of 0 to 2.5Volts. Therefore, scale the values sent to the DAC with the scale constant that converts the -10 to +10 range to 0 to 2.5. Send these scaled values to DAC channels 1 & 2.

Use the function PWM_out() to command a PWM channel. For example, PWM_out(EPWM4, 5.0) would produce a 75% duty cycle square wave on PWM channel 4’s output. To write to both DAC channels use the function writeDAC7564(float dac1, float dac2) with the two parameters variables floats with values in the range 0 and 2.5.

c. As in previous lab, use a long integer variable to keep a count of the number of milliseconds elapsed. Using this “count” variable, print the elapsed time in milliseconds to the text LCD screen every 100ms. On the TMS320F28335 the function to write to the text LCD screen is

LCDPrintfLine(unsigned char line, char *format, ...). This function, as with the LCDPrintfLine function on the OMAPL138 processor, works the same as the ANSI C printf function except for the added first parameter that states which LCD line the text is to be printed.

Additionally, print to the second LCD line the radian measurements from optical encoder 1 and 2. Only print one decimal precision of these floating-point values so you can see the full values on the single 20-character line of the LCD.

d. Also, every 100ms, toggle on and off one of the LEDs of the TMS320F28335 control card. The LED is connected to pin GPIO34. To control the state of this pin (and whether the LED is on or off) use the bit field structure GpioDataRegs. To simply toggle the bit on and off as the assignment asks, use the toggle command

GpioDataRegs.GPIOBTOGGLE.bit.GPIO34 = 1;
to clear the IO bit use (FYI)
GpioDataRegs.GPIOBCLEAR.bit.GPIO34 = 1;
and to set the IO bit use (FYI)
3. If you have not already, connect the JTAG debugger to the TMS320F28335’s JTAG connector which is at the top left corner of the circuit board close to the “TMS320F28335 Control Card”.

4. Build and run your application to confirm it is working. NOTE that when you download code from CCS through the JTAG to the TMS320F28335 you are flashing your code to the TMS320F28335’s flash memory. This is the reason that it takes longer for the JTAG to download code to this processor. This is exactly the same thing that is happening when you are programming your microcontroller for your homework assignments. This means that after you have downloaded and debugged your code, the code is sitting in the flash ready to be run the next time you turn off and on the robot.

5. Verify your program by ‘scoping’ the DAC outputs. While moving the wheels on the robot, the DAC voltage should vary between 0 and +2.5 volts.

6. Also verify your program by scoping the PWM outputs. While moving the wheels on the robot, the PWM duty cycle should vary from 0% to 100%.

7. Disconnect CCS from the robot and carefully remove the JTAG from the robot board. Test that indeed your program has been stored in the flash of the TMS320F28335 by turning off and on the robot and see that your program begins running again.

8. Set the robot on the ground and find the conversion factor from motor angle to length of a floor tile (1ft). Do this by lining up the robot’s front wheels with the edge of a floor tile, turn off then on the robot to zero the encoder readings, push the robot three or four tiles and record the number of radians the motor’s shaft has turned. This is easily done by printing the encoder count to the LCD screen. Record this radians-to-tile ratio as you will need it for calculations later in this lab and in the remainder of the class.

9. Check your ratio with your instructor before proceeding to next exercise.

10. Set the robot back on its stand on the table and reconnect the JTAG to the TMS320F28335 JTAG port.

**Exercise 2: Basic Feedback Calculations**

Partners switch places to allow other partner to program. For this lab we will attach a third optical encoder to Enc3_rad’s input connector that you will be able to hold in your hand and use like a joystick to dial in an input to the robot. We will also no longer be using PWM outputs 4 and 5 which were just attached to the banana jacks for connecting to the oscilloscope.

11. Continue working with the same code as exercise #1

12. Create two global float variables \( u_1, u_2 \), to be used as the control outputs to PWM channels 1 and 2. Modify your previous code so that every 1ms the third optical encoder, which you can manually turn, is used as the command to drive the robot’s two motors with the same value. This way you will have manual control of the robot moving forward and backward at different speeds. Perform the following:

   a. Set both \( u_1 \) and \( u_2 \) equal to \( \text{Enc3\_rad} \). *NOTE, in later labs \( u_1 \) and \( u_2 \) will be assigned values calculated by a control algorithm.*

   b. Limit \( u_1 \) and \( u_2 \) to a value between \(-10\) and \(10\).

   c. Command \( \text{EPWM1} \) and \( \text{EPWM2} \) to the value of \( u_1 \) and \( u_2 \) using the \( \text{PWM\_out()} \) function.
13. Using the backwards difference rule \( v = (p_{\text{current}} - p_{\text{old}})/0.001 \) as a discrete approximation to the derivative, calculate the linear velocity corresponding to each motor in \text{tiles/second}\ from the past and present motor position measurements, \( p_{\text{old}} \) and \( p_{\text{current}} \) respectively.

14. Print the velocity values of \( v_1 \) and \( v_2 \) on the first line, and the reading from encoder 3 on the second line of the LCD screen every 100 ms. You have to limit the accuracy of your floating-point number printed in order for all the information to be displayed completely on the 20-character LCD line.

15. Make sure that the robot is on the bench stand with wheels OFF the ground. Enable the PWM amplifier when running this code so the motors will spin. The amplifier is enabled by flipping up the switch on the robot’s front amplifier board. When the motors start spinning check the direction they are spinning. Given a positive input, both the motors should be spinning so they drive the robot forward. \textit{The back end of the robot is the end the battery can be inserted.} If both the motors do not move in the forward direction when sending a positive control effort, change the sign of the control effort for the motor that is spinning in the wrong direction. The best place to add this negative sign is in front of the parameter passed to the appropriate \texttt{PWM\_out()} function. Do not change the sign of the control variable, e.g., if you think motor 1 is spinning the wrong way given a positive control input, add the sign change to the parameters of \texttt{PWM\_out()} and do not do something like \( u_1 = -u_1 \); The sign change is necessary because the motors under the robot are mounted in opposite directions.

When you have both motors spinning in the positive direction given a positive PWM input, check the signs of the two velocity readings. If either of the velocities is not positive you will need to change the sign of the encoder measurement. The easiest way to do this is before you use either the \texttt{Enc1\_rad} value or \texttt{Enc2\_rad} value, change the variable’s sign with the command \( \texttt{Enc1\_rad = -Enc1\_rad} \). Of course, change this statement using \texttt{Enc2\_rad} if it is the variable with the wrong sign. Verify that your velocity (tiles/sec) calculation changes if you put a load on the wheels (by adding resistance to motion with your hand).

**Exercise 3: Friction Compensation**

In this final exercise you are going to implement a friction compensation algorithm to reduce the effects of the motor’s friction on your system. Your task will be to identify the friction acting on the motor. To accomplish this, you will produce a plot of \texttt{applied motor control effort (or input)} (on the y axis) vs. \texttt{motor velocity} (tiles/s) (on the x axis). You will first record 10 or so data points split between positive and negative values. Then by making a plot of your data, you can determine both the viscous and coulomb friction of the motors in both the positive and negative directions.

16. Using the third encoder to change the open loop command to the motors, record the steady state tile/sec velocity of the robot driving on the floor at 10 different control inputs. Be sure your measurements make sense… if you read 1 tile per second from the LCD, then the robot should actually be moving approximately 1 tile every second. If the robot is driving too fast to measure the values, you will obviously have to use a smaller magnitude of control input. Be sure to measure both positive and negative velocity values (robot moving forward AND backward).

17. Produce a plot of control input vs. speed in MATLAB to show to your instructor.

18. Using two separate regression fits, one for positive velocities and one for negative velocities, find the two lines that best fit your measured data points in the positive and negative velocity regimes. Ask your instructor for help if you don’t know how to quickly do a regression in MATLAB. From the best-fit lines, note the y intercepts. These are
your values for positive and negative coulomb friction. We will label these values $C_{pos}$ and $C_{neg}$ (note $C_{neg}$ will be a negative value). The slopes of the two lines are your values for positive and negative viscous friction. We will label these values $V_{pos}$ and $V_{neg}$.

19. With these identified parameters, add the following friction compensation algorithm to your code (assuming your motor one command variable is called “$u_1$”):

   ```
   if (v1> 0.0) {
       u1 = u1 + Vpos*v1 + Cpos;
   }
   else {
       u1 = u1 + Vneg*v1 + Cneg;
   }
   ```

   And perform similar instructions for $u_2$

20. As a starting point use 100% (in future labs we will only use 60 - 80% of the identified values) of the values identified for $V_{pos}$, $V_{neg}$, $C_{pos}$, and $C_{neg}$. Compile your code.

21. Test your friction compensation code by pushing the robot when it is on the floor. Your robot should roll for a long distance if the friction compensation is working well. In your code make sure that only the friction compensation is activated for this section. The easiest way to do this is to assign $u_1$ and $u_2$ to zero each time in your code section instead of assigning $u_1$ and $u_2$ the value of $Enc3_{rad}$. You may need to hand tune your friction gains slightly to get the robot to roll for a good distance when you push it. Demonstrate your working code to your instructor.

Lab Check Off:

1. Demonstrate your first application that reads optical encoder inputs 1 & 2 and echoes the radian values to PWM channels 3 & 4 and DAC channels 1 & 2. Scope the outputs to demonstrate your code is working.

2. Demonstrate your second application that drives the motor with an open loop PWM output and prints this control effort and both motor’s speed to the LCD.

3. Show your instructor the plot of the motor control vs. velocity friction curve and demonstrate your working friction compensation code by pushing the robot back and forth between partners.