

Mechatronics Laboratory Assignment 3

Introduction to I/O with the Daughter Card

Recommended Due Date: By your lab time the week of February 16th

Possible Points: If checked off before your lab time the week of Feb. 23rd ... 10 points

If checked off after your lab time the week of Feb. 23rd and before your lab time the week of Mar. 2nd ... 7 points

If checked off after your lab time the week of Mar. 2nd and before your lab time the week of Mar. 9th ... 5 points

If work not finished before your lab time the week of Mar. 9th ... 0 points

Goals for this Lab Assignment:

1. Learn functions to read Optical Encoder Channels 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.

DSP/BIOS Objects Used:

PRD

Daughter Card Library Functions Used:

Init_encoders, read_encoders, Init_pwm, out_PWM, writeDAC7724, Init_DAC7724, Init_UART1and2, LCDPrintfLine1, LCDPrintfLine2, WirelessPrint

Prelab: None... but you should be able to start (and possibly complete) the assignment at the end of this lab for lab 5 AND the prelab for Lab 4. Both are extensive.

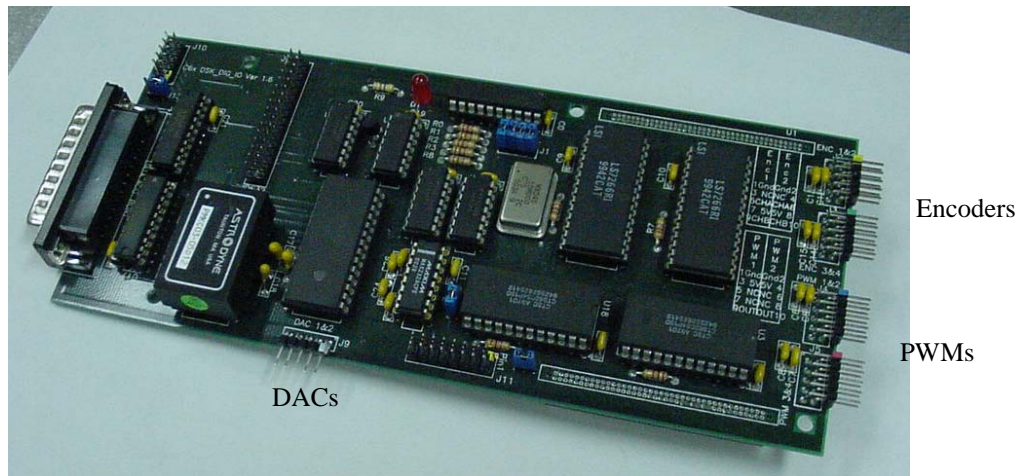


Figure 1: The C6XDSK I/O Daughter Card

Laboratory Exercise

Instead of doing something new with the DSP/BIOS kernel, today we begin to focus on Mechatronics, specifically using the DSP to measure the outside world and react to those measurements as a mechanical/electrical system. The DSP on your robot has an attached I/O daughter card that is piggybacked on the C6713DSK board. This board gives the DSK the following I/O, all of which can be sampled at least 10000 times a second:

- 4 Channels of Optical Encoder Input.
- 4 Channels of PWM Output.
- 4 Channels of +/- 10V DAC Output.
- 4 Channels of +/- 10V ADC Input (a +/- 300 deg/s rate gyro is attached to ADC 3)
- 8 Digital Inputs.
- 8 Digital Outputs.

Additionally the DSP interfaces to some slower sensors. Their sample rates are on the order of 5 to 10 times a second.

- Digital Compass with angular resolution of 2.5 degrees.
- Infrared Distance sensors with range of about 1/2 meter, resolution of about 1 cm
- Ultrasonic Distance sensors with range of 2 meters (when mounted that low to the ground on the robot), resolution of about 5 cm
- An I2C sensor bus easily used for additional sensors

Figure 2 is a simplified schematic of the daughter card. The full schematic along with the PWM amplifier board schematics can be found at coe.csl.ece.uiuc.edu/ge423/datasheets/c6713_dgio.pdf. The functions that interface the DSP to this board are found in the file `c6xdskdigio.c`. Use this source file along with the **DaughterCard_LibFuncs.pdf** document to help you write your code with these functions.

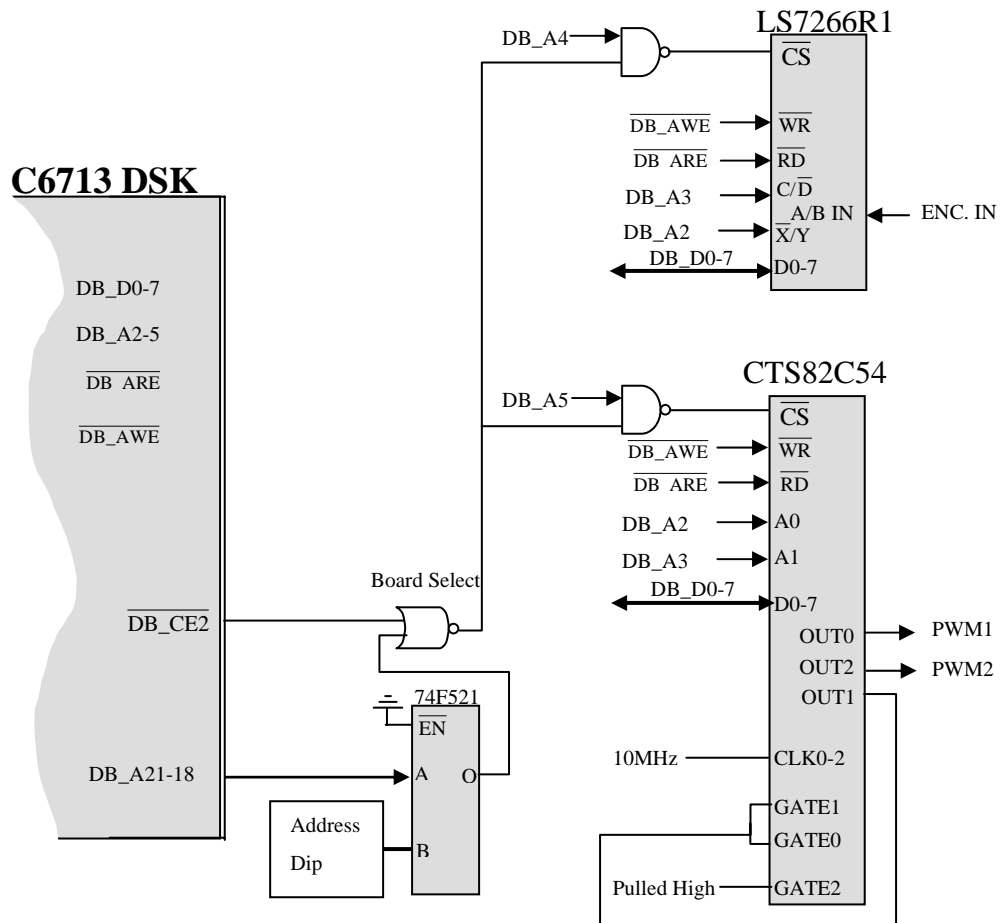


Figure 2: C6XDSK I/O Daughter Card Partial Schematic

You should now be very familiar with the PRD object, so we will no longer discuss the steps for creating the exercises for this lab. Instead, we are simply going to list the requirements for the exercises and expect the student to develop the appropriate code.

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 robot suicide, i.e. the robot may suddenly drive itself off the bench. If a student catastrophically breaks their robot, both the TA'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, and add a PRD object, **PRD_basiciopr**, to call a function **basiciopr** every 1 ms
2. Write the function, **basiciopr**, and associate it with the PRD object to do the following (you will need to read the handout on external functions to do some of these steps):
 - a. Every 1 ms, sample optical encoder channels 1 & 2 from encoder chip 1. You will have to specify encoder options for a standard gear motor, and you will see how to set these options by reading the appropriate include file. Note that this function returns the radian value of the motor.
 - b. Every 1 ms, send the radian value of encoder channels 1 & 2 to the PWM output channels 1 & 2 on chip 2 (or also called PWM channels 3 and 4). Send the same radian values to the DAC output channels 1 & 2. *A coding note here. PWM chip 2 can either be setup to drive DC motors like the ones that power the robot or it can be configured to drive RC hobby servo motors (Lab 7). Most projects will include RC servo motors instead of extra DC motors. For this reason Project_Creator's generated code initializes PWM chip 2 for RC servo control. See in `main()` that the function `Init_RCservo()` is called. For this lab we are going to want to change this. So comment out the line `Init_RCservo()` and replace it with the line `Init_pwm(2)`. The only reason we are having you do this is so that you can see on the oscilloscope what the PWM signal looks like when driving the DC Motor.*
 - c. Print out the integer time every 100 ms on the first line of the LCD screen.
 - d. Display the radian measurements from encoders 1 and 2 on the second line. Only print one decimal of the floating-point value.
3. Build and run your application to confirm it is working.
4. Verify your program by 'scoping' the DAC outputs. While moving the wheels on the robot, the DAC voltage should vary between -10 and +10 volts.
5. 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%.
6. Using instructions below, flash the code on the DSP so it can be run autonomously.
 - a. Keep your current CCS session running
 - b. Open up a TI Environment DOS box and change your directory to the Flash directory in your project
 - c. Type **flashit** at the command prompt and wait approximately 20-30 seconds for your program to flash.

- d. Exit CCS.
 - e. Either hit the Reset button on the DSK or power the DSP on/off/on. Your code should now run automatically when the DSP is powered on. If it works, your FLASH program is complete.
7. Set the robot on the ground and measure the number of radians on each of the wheels correspond to one tile unit as you push the robot. 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.
 8. Check your ratio with the TA before proceeding to next exercise.

Exercise 2: Basic Feedback Calculations

Partners switch places to allow other partner to program.

9. Use same code as exercise #1 (you may want to copy Part 1 code to a backup directory). Keep sample period at 1 ms.
10. Create four global float variables **u1, u2, enc3 and enc4**, to be used as the control output to PWM channels 1 and 2 on Chip 1 and optical encoder 3 and 4's reading. (Note: you will only be attaching one optical encoder sensor to the robot so only the variable **enc3** will have a valid sensor reading.) Modify your previous code so that every time interval it:
 - a. Assigns **enc3** with the position reading of the attached encoder (hereafter called encoder 3) connected to channel 1 on encoder chip 2,
 - b. Sets **u1** and **u2** equal to **enc3**'s value, in later labs **u1** and **u2** will be assigned values calculated by an control algorithm,
 - c. limits **u1** and **u2** to a voltage between -10 and 10, and
 - d. sends these values to the PWM channels 1 and 2 Chip 1.
11. 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 tiles/second from the past and present motor position measurements, p_{old} and p_{current} respectively. There is a problem that we have to address in the calculation of this velocity from encoder position readings. The problem is that the encoder interface chip that senses the optical encoder's position only has a finite number that it can count up to. It is a 24 bit counter value so the maximum number is $2^{24}-1$. In other words there is going to be a point when the encoder count must roll over back to zero. At that point our velocity calculation would be huge and cause problems for our control algorithm. At the beginning of lab we will discuss a way to get around this problem.
12. Print the velocity values of **v1** and **v2** on the first line, and the reading from encoder 3 second line of the LCD screen every 50 ms. You have to limit the accuracy of your floating-point number printing in order for them to display completely on the LCD screen.
13. Demo your code to the TA before continuing.
14. 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 motor will spin. The amp is enabled by flipping the switch on the front amp board. If the motors do not move in the forward direction when you send a positive voltage, change the sign on your control effort right before you send it to the PWM. If the motors do not read a positive velocity as the robot drives

forward, change the signs on your encoder readings appropriately. Verify that your tile velocity 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 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 **motor velocity** (tiles/s) (on the x axis) vs. **applied motor control effort (or input)** (on the y axis). You will first record 10 or so data points split between positive and negative voltages. Then by making a plot of your data, you can determine both the viscous and coulomb friction of the motor in both the positive and negative directions.

- Using the encoder to change speeds for each reading, record the steady state tile velocity of the robot driving on the floor at 10 different control inputs (and velocities). 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 a value, 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).
- Produce a plot of speed vs. control input in MATLAB to show the TA.
- 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 TA for help if you don't know how to quickly do a regression in MATLAB. From the best-fit lines, note the 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} .
- With these identified parameters add the following friction compensation algorithm to your code (assuming you were outputting a variable before called $u1$):

```
if (v1> 0.0) {
    u1 = u1 + Vpos*v1 + Cpos;
}
else {
    u1 = u1 + Vneg*v1 + Cneg;
}
And perform similar instructions for u2
```

- 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.
- 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 $u1$ and $u2$ to zero each time in the period function instead of assigning $u1$ and $u2$ the value of **enc3**. You may need to hand tune your friction gains slightly to get the robot to roll for a good distance when you push it. Demo your code to the TA.

Lab Check Off:

- 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 the motor's calculated speed to the LCD.
3. Show your TA the plots of the motor velocity/control input curve for the robot, and demonstrate a robot with working friction compensation and encoder roll over corrected.

Extended Prelab/Final Check Off Assignment for Lab 5:

In remaining labs, you will be driving the robot around the room, and starting on Lab 5 we will ask that you display the estimated position of the robot in a Visual Basic application. Every 0.25 seconds, your DSP will send the PC the following information: its x-location (in tiles), its y-location (in tiles), its velocity (in tiles/second), and its orientation (in radians) as measured by the rate gyro. Additional sensor readings of your choice may also be sent. On part of your application window, the VB routine should then plot the position and orientation of the robot. As your robot moves around on the floor your depiction of the robot should move around in your VB application. Because this is a challenging task, you should start thinking and reading about it now. Also, the second half of Lab 4 steps you through drawing a circle in VB that moves in a PictureBox given input from the Robot's ADCs. So by the end of Lab 4 you should have a good idea on how your VB application is going to work. Below are steps to guide your development of this VB application. Each lab assignment starting with Lab 5 will ask you to continue adding functionality to your VB application and demonstrate it working at each of your lab check-offs.

1. Create a VB program whose window uses most of the screen. In this window create an area that will represent the robot course. One way to do this is to use a picture box object. In the picture box draw grid lines (using the Line method) for a 20 by 20 tile course. Use a circle to indicate where the robot is located in the course and draw a line to indicate the direction and velocity of the robot. The length of the line will indicate the velocity. Find in VB's Help (See VB's function reference help section) the following to help you with these tasks: PictureBox Autoredraw property, PictureBox ScaleTop, ScaleHeight, ScaleLeft and ScaleWidth property, a shape object, a line object, PictureBox Cls, Line and Circle Methods.
2. You will be able to pick a starting location and orientation for you robot in the course.
3. Using the VB code from Lab 2 receive the X, Y, angle, velocity data sent from the DSP over the wireless serial interface. Send your variables in a string just like we sent the intime variable to VB in Lab 2. This way you can determine the number of decimal places to send for each of your variables. A little more code is needed at the VB end for this method to parse the string into your individual variables. VB has a function called "Split" that can parse a delimited string into separate strings. After the string has been parsed you can use the VB function "Val" to convert the separate strings into a number. VB's help shows an example of using the Split function. (See VB's function reference help section). Later, we will be adding sensor readings to the robot; these will be displayed in text boxes in your VB application.

This 'dead reckoning' method of navigation we are using has errors that propagate the farther you travel (i.e. the errors are integrated). While completing the remaining labs, start thinking about how you might use various sensor readings to measure course features that will help correct your robot position estimate. Examples include measuring wall distances to correct (x, y) positions, using visual landmarks to correct angular errors, etc.