

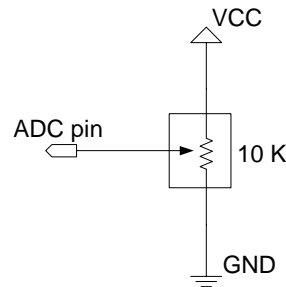
GE 423 Mechatronics Homework Assignment #2
Spring 2009, Due In Lecture February 25th. The Microcontroller Demonstration Check-Offs for Questions 4, 5, 6 and 7 are due by 3PM Tuesday February 24th.
Most answers should be typed. Graphs, etc. can be hand drawn if you wish.

1. Read Chapters 6-8 in “Teach Yourself C”. Read Section 10.5.3 in “Introduction to Mechatronics”. Read the article “Serial Protocols Compared” (<http://www.embedded.com/showArticle.jhtml?articleID=9900637>) or at (<http://coecsl.ece.uiuc.edu/ge423/datasheets/SerialProtocolsCompared.pdf>). Look at Wikipedia for information on both I²C and Serial Peripheral Interface (SPI). For I²C, the C6713 I²C Module Reference Guide http://coecsl.ece.uiuc.edu/ge423/datasheets/C6713Ref_Guides/I2CRefGuide.pdf has good information and diagrams.
2. Explain the source code given in Lab 2 for the Visual Basic function `SerialCom1_OnComm` (page 2). Use both a flow chart and a written explanation to describe what this function is doing. The key to understanding this piece of code is to know that the `RThreshold` property for the VB serial port is set to 1. This means that the `SerialCom1_OnComm` function will be called automatically when there is *at least one character* in the PC’s serial port buffer. The *at least one character* is important. There could be one character in the buffer or there could be more than one. So when the `SerialCom1.Input` variable is read in the code, the code needs to handle the input as a string of characters. **This is probably not the full message sent from the DSP!** The string of characters read will most likely be just part of the full message you are sending from the DSP. Also note that the variable `strNextString` is a global variable and therefore does not lose its data when the function exits.
3. Explain the source code given in Lab 2 for the function `UART1ReceiveCharTask` (page 5). As in question 2 draw a flow chart and write an explanation of this code. Remember that this code runs on the DSP and this task function gets “woken up” each time an individual character is received. That is different from the VB function above. The DSP is much faster and it will process each character as it arrives.

Questions 4 through 7 below build on each other. So just use the project creator once to give you a starter shell for question 4. After that, add the needed code to the previous problem’s code. If the question asks you to remove portions of the code from the previous question leave the code but comment it out (`/* old code */`). You will be graded on these commented sections. Demonstrate each assignment to your TA and print and hand in your completed code.

4. Create a new project using the eZ430 project creator http://coecsl.ece.uiuc.edu/ge423/datasheets/MSP430Ref_Guides/exe/EZ430ProjectCreator.exe. Then for this question you only need to change the count mode of Timer A to UP count mode. Read section 12.2.3 of the MSP430 users guide for information http://coecsl.ece.uiuc.edu/ge423/datasheets/MSP430Ref_Guides/MSP430x2xx_usersguide.pdf. The default project given to you by the project creator sets Timer A in *continuous* count mode. In this mode the compare register CCR0 needs to be updated each timer interrupt so that your desired sample rate interrupts the MSP430. In UP count mode this is no longer necessary because this mode counts up to the CCR0 value, then resets the timer back to zero and starts counting up again. You are being asked to change to the UP count mode because in question 7 you will be asked to create a pulse width modulation (PWM) output and UP count mode is required for that. Make the necessary changes to your code and verify that your LED is still blinking at the same rate as the default code and you are still sending data over the serial port to your PC. See example file http://coecsl.ece.uiuc.edu/ge423/datasheets/MSP430Ref_Guides/Cexamples/c/msp430x20x3_ta_02.c for help.

5. Write code to read ADC channel A5 of the MSP430F2012. First you will need to solder one of your 10 Kohms potentiometers to the EZ430 Break-Out board. Pin 1 of the pot should be wired to GND and pin 3 should be wired to VCC (+3.3V). Pin 2 of the pot should be wired directly to P1.5(A5) MSP430F2012 pin.

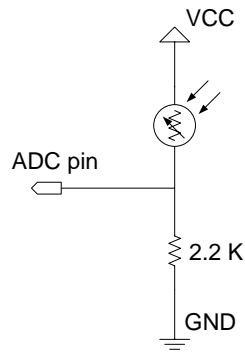


Modify your default project creator C-file to sample the voltage of pin A5 and print its integer value to the UART serial port every 0.25 (2400/9600) seconds. Make sure not to read the sample ADC value until the ADC interrupt has occurred. Hand in a print out of your code and demonstrate it working to your TA. Your code should be commented well enough that a beginner working with this microcontroller can understand what each added instruction is accomplishing. The example http://coecsl.ece.uiuc.edu/ge423/datasheets/MSP430Ref_Guides/Cexamples/c/msp430x20x2_adc10_01.c gives you a start on using the ADC10 peripheral of the MSP430F2012. Note: This example takes advantage of the power saving features of the 2012 which we are not interested in using for this problem (i.e. the CPUOFF setting). You will also need to read the ADC10 section of the 2012's users guide http://coecsl.ece.uiuc.edu/ge423/datasheets/MSP430Ref_Guides/MSP430x2xx_usersguide.pdf to become familiar with the ADC and its registers. Make sure your ADC is set as follows in your `main()` function. Note that a number of these are the default settings of the ADC10 and therefore no code is needed to set the ADC in that mode.

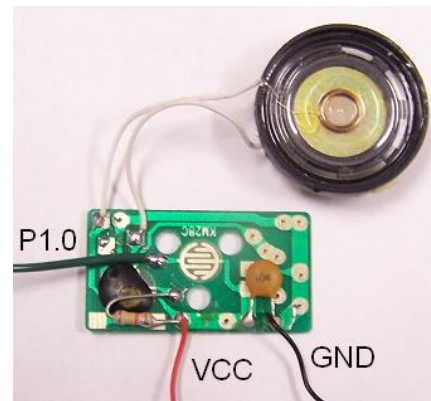
- a. $V_{r+} = VCC$ and $V_{r-} = V_{ss}(GND)$
- b. Sample-and-hold time $8 \times ADC10CLKS$
- c. Fast mode (up to 200K sample per second)
- d. Reference output off
- e. Reference buffer on continuously
- f. Multiple sample and conversion disabled (Single conversion mode)
- g. Reference generator off.
- h. Interrupt enabled
- i. Select input channel A5
- j. Sample-and-hold source = ADC10SC (software starts conversion)
- k. Straight binary Data format
- l. Clock divider--- try 1, 4 and 8 and see if you can notice any difference in signal noise just by watching the values printing in HyperTerminal.
- m. Clock source = ADC10OSC
- n. Conversion sequence = Single-channel-single-conversion
- o. Enable A5 as the ADC channel.

What is the minimum and maximum integer ADC value you would expect to see printed to HyperTerminal? What bit in an ADC control register could you poll on (or monitor) to determine that an ADC conversion was complete if you did not use the ADC interrupt?

6. Make Santa say "HO HO HO" when the lights are turned off. Here you will wire/solder the photo-resistor to your EZ430 Break-Out board. A photo-resistor is a device that changes resistance with respect to light intensity. If you wire the photo-resistor in series with a resistor that has a similar value as the minimum photo-resistor resistance (2.2K ohms should probably work), you can measure a voltage that varies with light intensity. The node in the circuit you should measure is the point where the resistor and photo-resistor are connected. See the figure below:



This circuit is called a voltage divider. One end of the voltage divider should be connected to VCC (+3.3V) and the other end to GND. The middle point is then wired to an ADC input and measured. For this assignment wire this point to ADC channel A6. Modify problem #5's code to sample A6 instead of A5 and print the value in HyperTerminal every 0.25 seconds. As a final step, wire the Santa voice box to digital output P1.0. This pin will emulate the pressing of the push button on the "HO HO HO" device, i.e. when P1.0 is high this will activate Santa. Use the following picture as a guide on

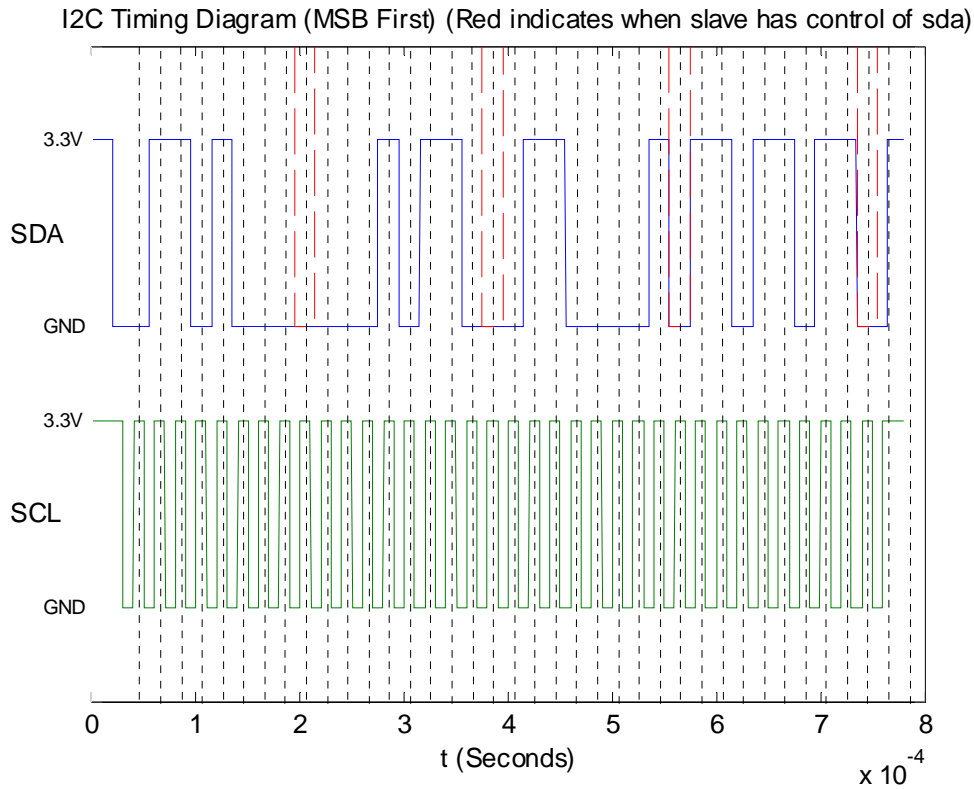


how to wire the "HO HO HO" circuit to your EZ430 breakout board.

A sample wiring of this entire circuit will be kept in lab as a reference. Modify your code to monitor A6's voltage value to determine when the lights have been turned off. When the lights have been turned off for 2 seconds make Santa say "HO HO HO." Your program should be able to continue working with repeated on/off's of the lights. Hand in a printout of your commented code and demonstrate it working to your TA.

7. Add one more feature to your photoresistor/Santa Claus circuit. Continuously vary the light intensity of LED 3 as a function of the photoresistor's light intensity read by ADC channel 6. The LED's light intensity can be changed by changing the duty cycle (PWM signal) of the logic pin controlling the LED. To do this change the functionality of the P1.2 pin to the TA1 mode. Use the P1SEL register to change this mode. See pages 58-67 in the MSP430F2012's datasheet http://coecsl.ece.uiuc.edu/ge423/datasheets/MSP430Ref_Guides/MSP430F2012_DataSheet.pdf for help in setting up P1.2 to its alternate function "TA1" (compare output or also called PWM output). Read section 12.2.5 of the MSP430 users guide for help on setting up the output mode http://coecsl.ece.uiuc.edu/ge423/datasheets/MSP430Ref_Guides/MSP430x2xx_usersguide.pdf. The C example http://coecsl.ece.uiuc.edu/ge423/datasheets/MSP430Ref_Guides/Cexamples/c/msp430x20x3_ta_16.c gives a little bit of help on how to setup the timer registers initially. Your PWM's carrier frequency is controlled by the CCR0 register. Leave this at the 9600Hz value needed for the serial port. CCR1's value determines the percentage of duty cycle. Once the timer registers are setup correctly, simply change the value of CCR1 as a function of the value received from ADC channel 6.

8. The plot below is the timing diagram for the transmission of data over an I2C serial port. What I2C address is the data being sent to? What data is being sent? Approximate the baud rate (bits/sec) from the plot.



9. Draw the timing diagram for the SPI serial port of our TMS320C6713 DSP set up to transmit and receive 8-bits at a time. The SPI registers have also been set so that CLKSTP=11b and CLKXP=0. See section 12.7 (pages 12-79 through 12-82) of the TMS320C6000 Peripherals Reference Guide (<http://coecsl.ece.uiuc.edu/ge423/datasheets/spru190d.pdf>) for details. Two values are sent, 88 then 201 while two values are received, 21 then 9.

F2012 Play-Time: (These items are not graded nor required)

1. Display the most significant bits (MSB) of ADC10's reading to the 4 LEDS. This would give you a quick macro display of the input voltage.
2. Start reading the data sheet for the Texas Instruments DAC TLV5606. A DAC changes a digital value to an analog voltage. This is useful for driving or controlling devices such as audio amplifiers and motor amplifiers. In HW 3 you will interface this DAC with the F2012. Ask your TA for a DAC chip if you would like to get started early on this assignment.