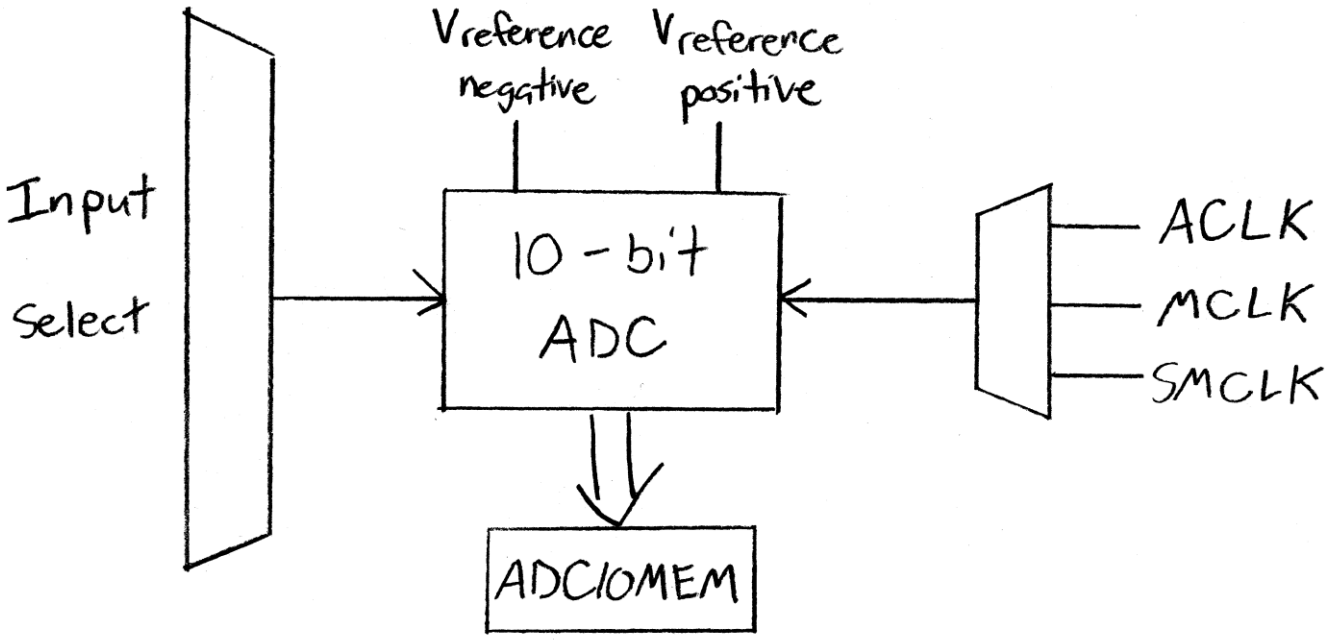
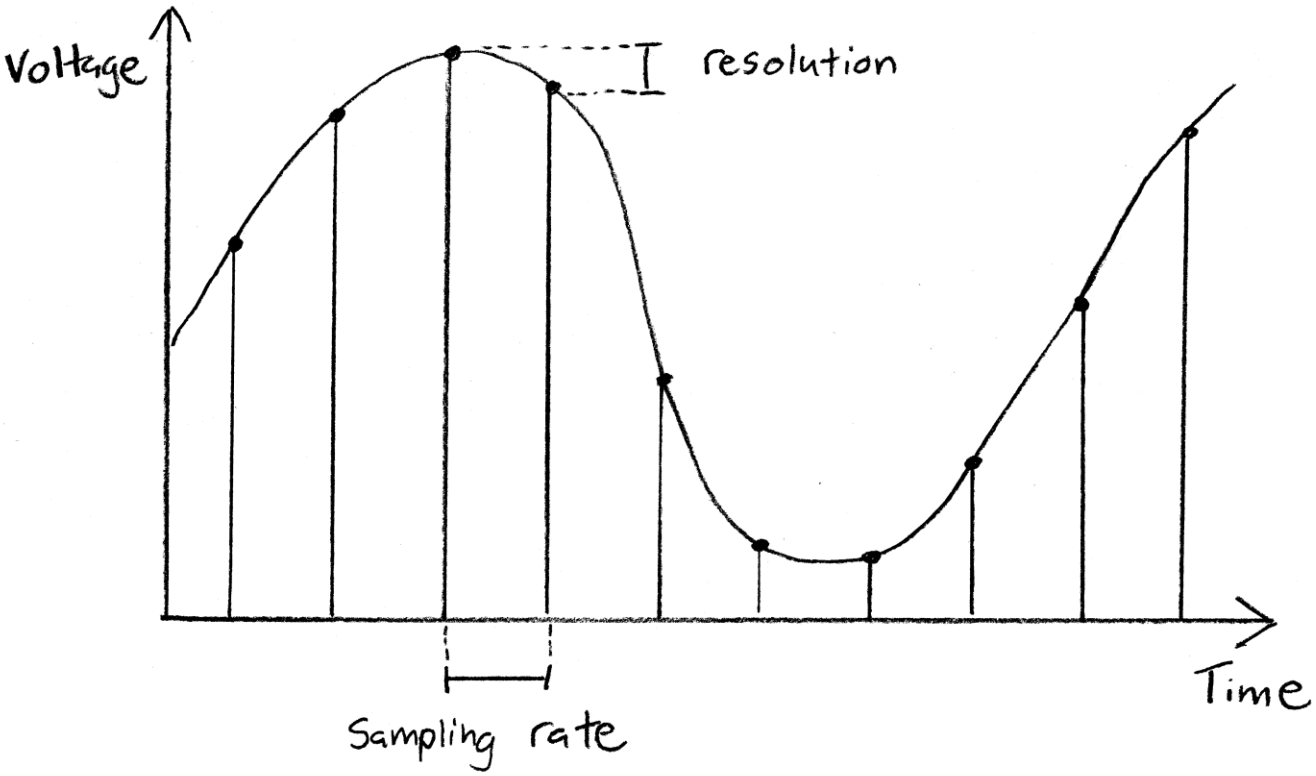
Lab IV: Digital Color Organ - Analog-to-Digital Converters, PWM Output, and Lattice Wave Digital Filters

A) MSP430 ADC10 Module

- 10-bit voltage resolution
- 200-Ksps sampling rate
- 8 Channels
  - Internal voltage and temperature
- Internal voltage references



B) Analog-to-Digital Conversion

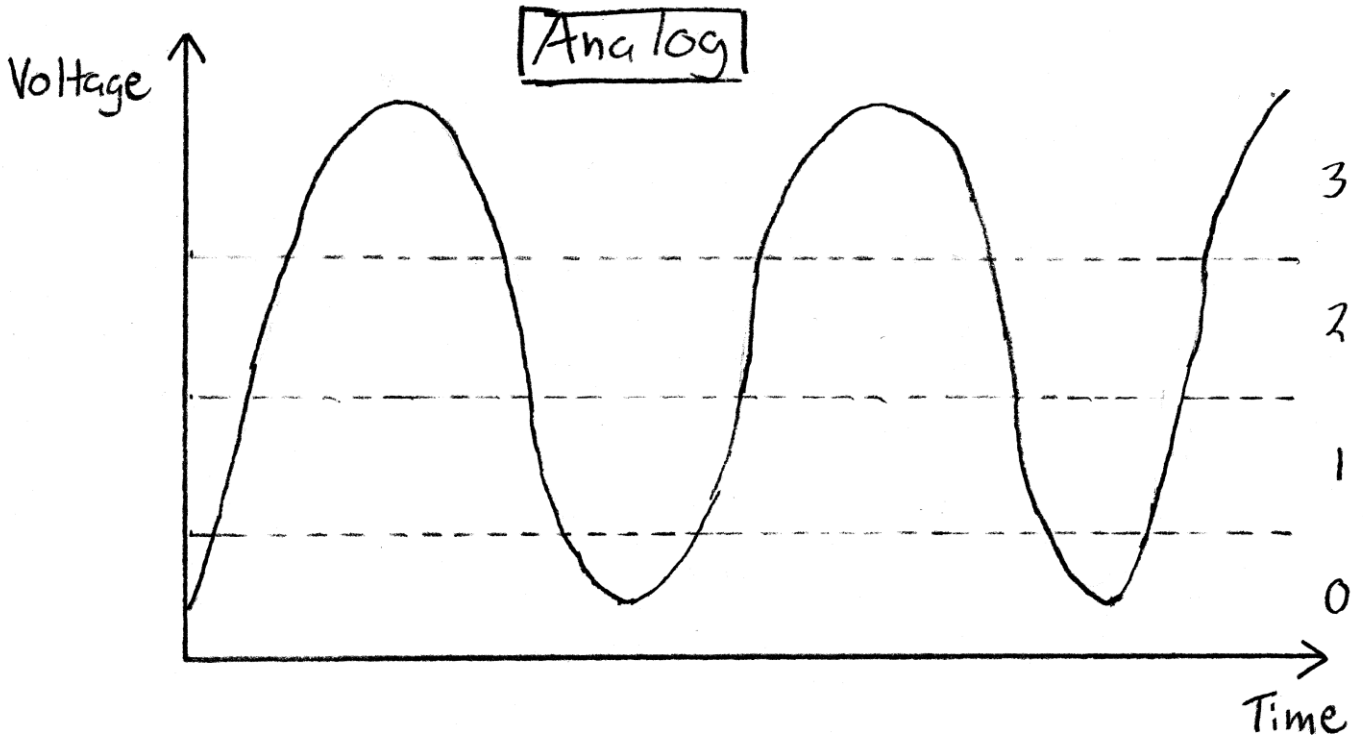


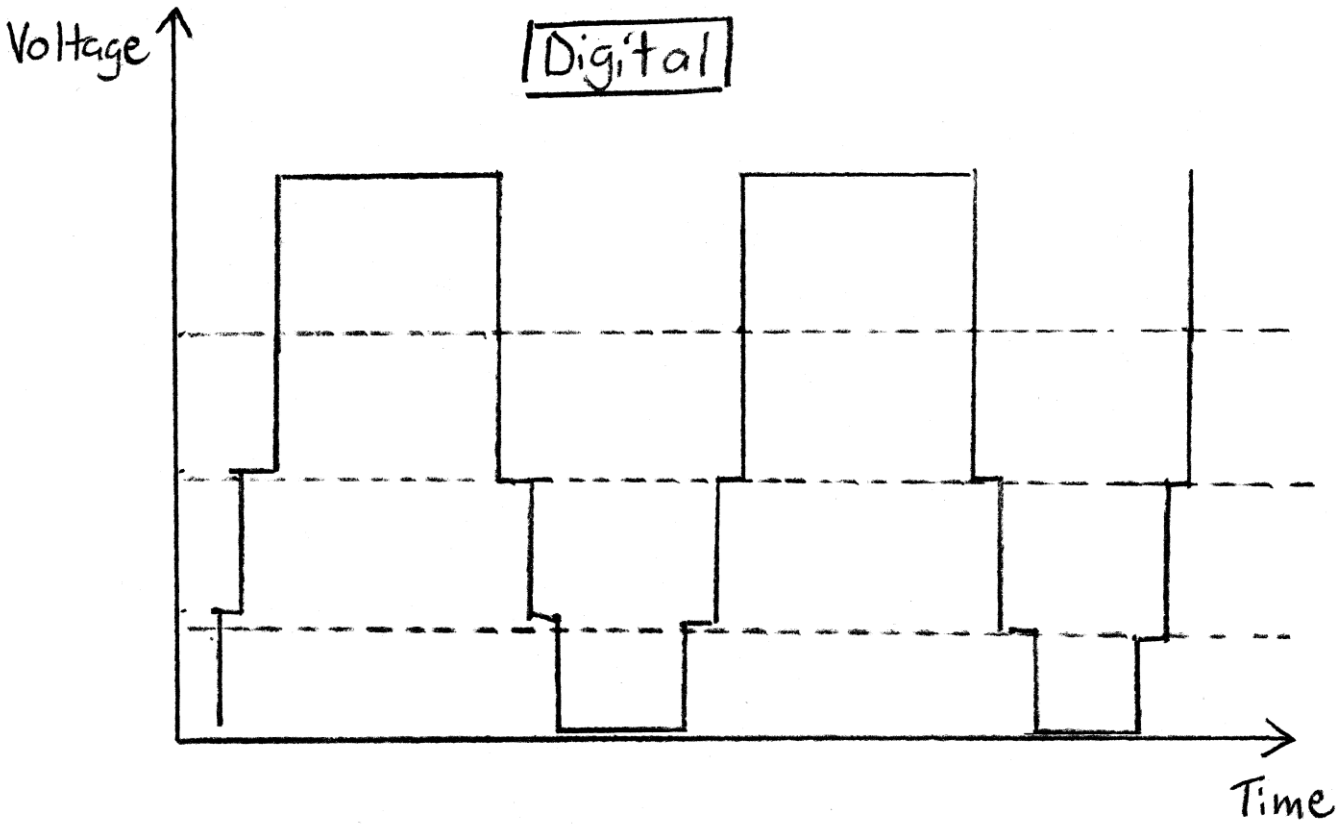
- Converts a continuous analog signal to a discrete digital signal.

- The sampling rate determines the maximum frequency you can resolve.

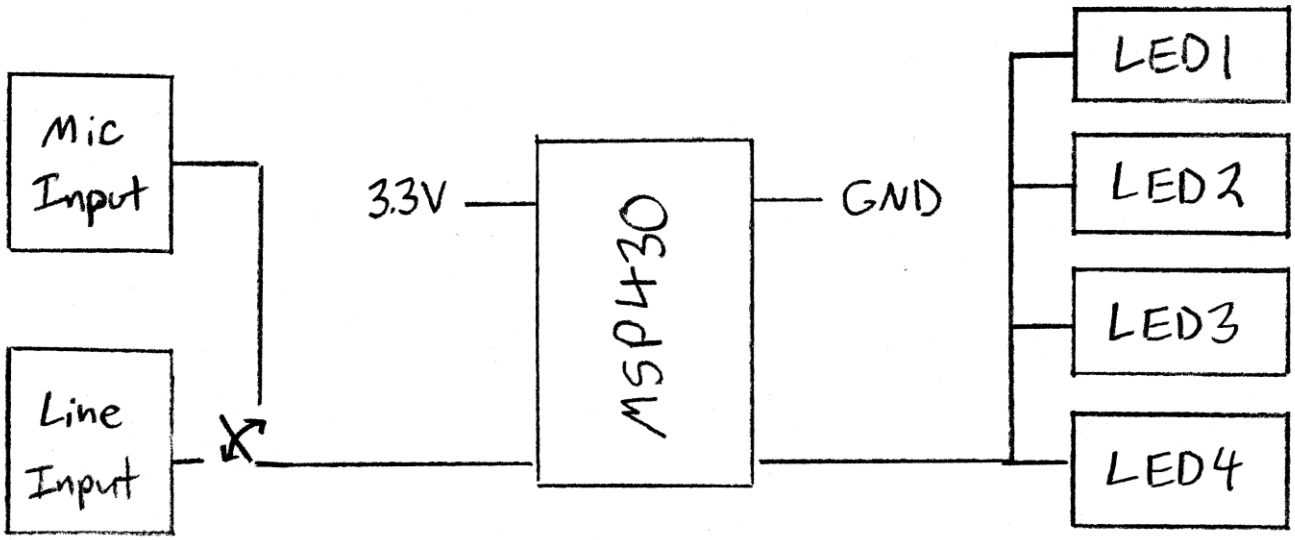
- the bit-depth determines the resolution, aka, the smallest voltage step.

1) 2-bit ADC Example





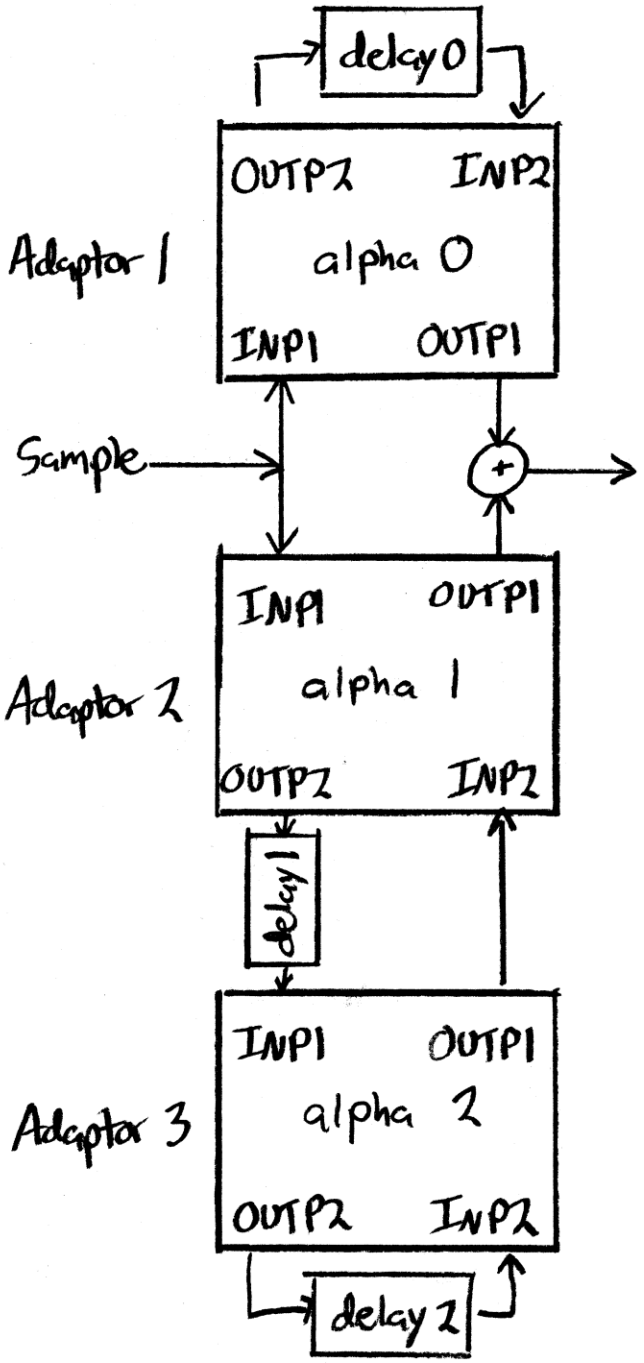
C) Single Channel Color Organ



- In this lab we will build a single channel color organ that modulates the brightness of all 4 LED banks in unison based on the amplitude of the input signal.

- Power and ground connections come from Lab I, and the analog input signal comes from Lab II.

D) Lattice Wave Digital Filters



$P1 = INP1 - INP2$

$OUTP2 = (\alpha_0 \times P1) + INP2$

$OUTP1 = OUTP2 - P1$

$P1 = INP2 - INP1$

$OUTP2 = (\alpha_1 \times P1) - INP2$

$OUTP1 = OUTP2 - P1$

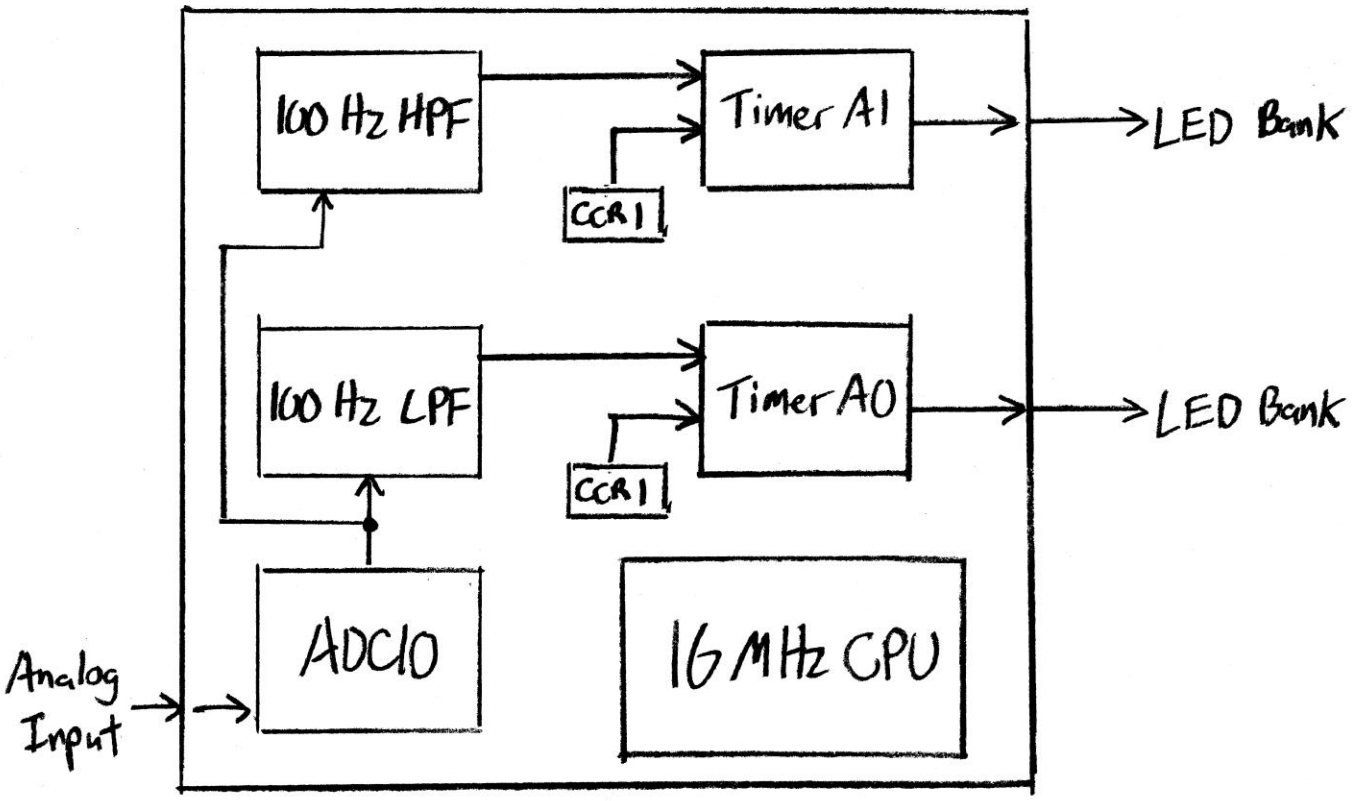
$P1 = INP1 - INP2$

$OUTP2 = (\alpha_2 \times P1) + INP2$

$OUTP1 = OUTP2 - P1$

- A lattice wave digital filter implements a set of analog filter specifications using only addition, subtraction, and multiplication operations.

E) Final Design



- The final design will use the analog input, sampled by the ADC10, filtered by two lattice wave digital filters (1 LPF, 1 HPF), to create two PWM outputs with two timers (A0, A1) and two capture compare registers that drive four LED bank outputs.

- This application uses almost all of the capibilities and processing power of the MSP430.

ECE 480L: SENIOR DESIGN SCHEDULED LAB

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

MICHIGAN STATE UNIVERSITY

I. TITLE: Lab IV: Digital Color Organ - Analog-to-Digital Converters, and Lattice Wave Digital Filters

II. PURPOSE:

A color organ is a device that takes an analog input, splits it up to different frequency bands, and controls the brightness of individual LEDs based on the amplitude of each frequency band. This lab will start by covering the concepts needed to implement these functions digitally, and create a basic single channel color organ. In this lab, you will use the MSP430's 10-Bit ADC to capture a sine wave, and use the amplitude to control the duty cycle of a single MSP430 timer, which will drive an LED. This program can be used to modulate the LED banks based on the amplitude of either the microphone amplifier or the line input summer. The final project will be to use a low pass filter and a high pass filter to create a two channel color organ.

The concepts covered are:

1. Setting up the ADC10 analog-to-digital converter.
2. Programming a lattice wave digital filter.

III. BACKGROUND MATERIAL:

See Lab Lecture Notes.
MSP430G2553 Data Sheet
MSP430 User's Guide
Texas Instruments SLAA331 Application Report

IV. EQUIPMENT REQUIRED:

1	Your own personal Bound Lab Notebook
1	Agilent Infiniium DSO-9064A Digital Storage Oscilloscope
4	Agilent N2873A 10:1 Miniature Passive Probes
1	Agilent 33250A or 33120A Function Generator

V. PARTS REQUIRED:

1	MSP-EXP430G2 LaunchPad kit
1	USB cable
5	Female-Male 6" jumper wires
1	BNC-to-Banana adapter
2	Banana-to-grabber wires

VI. LABORATORY PROCEDURE:

A) Configuring the MSP430 ADC10

1. We will start by setting up the ADC10 analog-to-digital converter to sample a sine wave, and display the result in Code Composer Studio. Chapter 22 of the MSP430 User's Guide contains all of the necessary information to fully utilize the features of the ADC10. It would be a good idea to browse through this chapter before continuing to the next step.

2. Start a new empty project in Code Composer Studio for the MSP430G2553. Label the project name **Lab IV - Part A**, and create a new source file called **lab4_parta.c**. If you have any difficulty doing this, refer to steps VI-A-1 through VI-A-6 of Lab III.

3. Copy and paste the following block of code starting at line 8 of your source file:

```
#include <msp430g2553.h>

void main(void)
{
        WDTCTL = WDTPW + WDTHOLD;

        while(1)
        {

        }
}
```

4. This will be our starting point from now on. Notice the 4 components contained within this block of code: a header file, a main function, a line to stop the watchdog timer, and an infinite while loop. Library files and variable definitions will be placed after the header line, initialization lines will be placed after the watchdog timer line, and the program code will go in the while loop. Interrupt functions can also be added, which are functions that only run when requested in the main function. These are normally placed at the end of the code after the main function.

5. Start by setting up the clock system. Use the following calibration coefficients to set the DCO to 16 MHz:

```
DCOCTL = CALDCO_16MHZ;
BCSCTL1 = CALBC1_16MHZ;
```

6. Next, we need to select which ADC10 channel we want to capture.

This is done using the ADC10CTL1 register. The MSP430G2553 has a total of 8 ADC channels. Refer to section 22.3.2 of the MSP430 users guide to find out how to setup the ADC10CTL1 register.

7. Notice that ADC10CTL1 is a 16-bit register instead of the 8-bit registers that we dealt with previously. This means significantly more initialization data is stored in the same register, which can make readability of the code difficult. Luckily there is a very easy way to write data to these registers without have to set individual bits.

8. The table for ADC10CTL1 lists 8 ADC channels, A0-A7. Use the pinout found in the MSP430G2553 datasheet to locate the physical pin that each ADC channel can be set to. Let's select the ADC channel A1 to be an input on P1.1. This can be done using the **INCH_1** variable to set bits 15-12 of the ADC10CTL1 register to select ADC channel A1. INCH_2 would select ADC channel A2, INCH_3 would select ADC channel A3, and so on. Using the **CONSEQ_2** variable will select repeat single channel mode.

9. Add the following line of code to your initialization section:

```
ADC10CTL1 = INCH_1 + CONSEQ_2;
```

10. By default, all of the analog input pins on the MSP430 are turned off to conserve power. We can use the ADC10AE0 register to enable the analog input pin A1. Add the following line of code after your **ADC10CTL1** initialization line:

```
ADC10AE0 = BIT1;
```

11. We are now ready to start the analog-to-digital conversion process. Refer to the section on the ADC10CTL0 register in the User's Guide.

12. First we need to configure the reference voltages for the ADC10. This is a 10-bit analog-to-digital converter, which means that the resulting digital value can be between 0 and 2^{10} - 1, which is 0 to 1023. The reference voltages will set the minimum and maximum voltages in this range; any voltage outside of this range will clip either high or low, and voltages inside this range will be set to a value between 0 and 1023. By default, the ADC10 uses the power supply voltage (Vcc) as the positive reference, and ground (Vss) as the negative reference. The power supply voltage can fluctuate, so it is best to use a more stable positive reference. The ADC10 has an internal reference that can be selected using the variable **SREF_1**. The variable **REF2_5V** will set this reference voltage to 2.5V

13. Both the reference voltage and the ADC10 must be turned on now. Since each of these variables only has one function, we can leave out the underscore and decimal number. Use the variable **REFON** to turn

on the reference voltage, and the variable **ADC10ON** to turn on the ADC10.

14. By default, the ADC10 will only take one sample, and then wait to be told to take another sample. We can use the variable **MSC** to tell the ADC10 to continue taking samples indefinitely.

15. The five variables from parts VI-A-12 through VI-A-14 can be implemented in one line of code. Copy and paste the following line of code in the initialization section of your code:

```
ADC10CTL0 = SREF_1 + REF2_5V + REFON + ADC10ON + MSC;
```

16. Before the analog-to-digital conversion process can be started, we need to allow time for the reference voltage to settle. Add this line of code:

```
_delay_cycles(5);
```

17. With everything set up, we can enable the ADC10 and start the conversion process. Copy and paste the following line of code into your program:

```
ADC10CTL0 |= ENC + ADC10SC;
```

18. The conversion result is stored in the variable **ADC10MEM**. Let's transfer this result to a temporary variable. Add the following line of code to the infinite while loop:

```
voltageRaw = ADC10MEM;
```

19. Remember that any new variable needs to be declared after the header file. Use the following variable definition:

```
float voltageRaw;
```

20. Turn on the Function Generator, and configure it to output a 0.1 Hz, 2.5 Vp-p Sine Wave with a 1.25 VDC offset. If you are using a different function generator than before, refer to the Lab 2 procedure for help. ***Remember to set High Z!***

21. The ADC pins on the MSP430 are very sensitive to over-voltage; before connecting the function generator to the MSP430, measure this sine wave on the Oscilloscope to make sure that the minimum voltage is 0V and the maximum voltage is 2.5V. Once you are satisfied that the sine wave is within range, connect the output of the function generator to the ADC channel A1 located at P1.1 using a BNC-Banana adaptor and two BNC-Grabber cables. ***Don't forget to make the Ground connection.***

22. Double check that everything looks good, and debug your code. If you get any errors, you will have to go back and fix them.

23. Once in the **Debug** perspective, double click on the variable **voltageRaw** in your code to select it, right click and select **Add Watch Expression**. Click **OK**, and navigate to the **Expressions** tab. This will show the value of **tempRaw** variable once the program execution has been stopped. Run the code, and press the yellow pause button labeled **Suspend**. You should see the value of **voltageRaw** update at this time.

24. Now we can set up the Debug mode to update the **voltageRaw** variable in real time. Select the voltageRaw line of your code, right click, and select **Breakpoint (Code Composer Studio) > Breakpoint**.

25. You should see a blue symbol added on the left hand side of your code. Right click on this symbol and select **Breakpoint Properties**. In the **Debugger Response > Action** line, change the value from **Remain Halted** to **Update View**.

26. Run your code again, and you should now see the **voltageRaw** variable updating a few times every second. Each time the variable changes, it is highlighted yellow.

27. What are the minimum and maximum values that you see? Does this make sense given how you set up the ADC10 reference voltages and the function generator?

28. Change the function type from Sine Wave to Square Wave and watch what happens. Does this make since? Set the Vp-p to zero, but leave the DC offset. What value do you see for **voltageRaw**?

29. Go back to the **CCS Debug** perspective and add one more line of code to your program to convert **voltageRaw** to a calibrated voltage reading. Use the information you gathered in the previous two steps as a guide. Remember what your voltage references and bit depth are.

30. Comment each line in your completed code for this section, and save your .c file.

B) Adding PWM

1. Start a new empty project in Code Composer Studio for the MSP430G2553. Label the project name **Lab IV - Part B**, and create a new source file called **lab4_partb.c**. Copy and paste your code from **Lab III - Part C**.

2. We can now add our code from Part A to use the ADC input to control the duty cycle of the PWM output. Copy and paste the 5 lines that you used to start the ADC10 conversion process from your Part A code into

the initialization section of your Part B code.

3. Now, instead of setting the value of TA0CCR1 manually, we can use the value in the ADC10MEM register to automatically set the duty cycle. Replace the single line of code to your infinite while loop with the following line of code:

```
TA0CCR1 = ADC10MEM;
```

4. Add the following lines of code to the initialization section to run the CPU at 16 MHz instead of 1 MHz:

```
DCOCTL = CALDCO_16MHZ;
BCSCTL1 = CALBC1_16MHZ;
```

5. **Debug** this new code. Setup the Function Generator to output a 1 Hz, 2.6 Vp-p Sine Wave (High Z) with a 1.25 VDC offset, and connect this to P1.1. Observe what happens to the brightness of the LED and the PWM signal on the scope when you change the frequency of the input, and also what happens when you switch between a sine wave and a square wave.

C) Color Organ Programming

1. Start a new empty project titled **Lab IV - Part C**, and create a source file called **lab4_partc.c**. Copy and paste your code from Part B into Part C.

2. We can now build a basic single channel color organ using our circuit from lab 2 as an input. This will require some conditioning of the voltageRaw sample before it is used to control the TA0CCR1 register.

3. Set the ADC10MEM register to a temp variable, voltageRaw. This will allow us to process voltageRaw before setting it equal to TA0CCR1 to control the Pulse Width Modulation output TA0.1.

4. The brightness of the LED is determined by the average value of the voltage that is applied. If you consider the maximum input signal (2.5V Vp-p with 1.25 V DC offset), and the minimum input signal (0V Vp-p with 1.25 V DC offset), both will have the same average value. You can subtract the DC offset in the code by subtracting the equivalent decimal value of 1.25V, given the value of the reference voltage and the bit depth of the ADC.

5. After you subtract the DC offset, the new signal will go negative, so you can fix this by taking the absolute value of the temporary variable.

6. Remember that you set the maximum brightness to occur at an

instantaneous voltage of 2.5V, but after subtracting 1.25V, the maximum voltage you can see now is only 1.25V. Maximum brightness needs to occur at 1.25V now, so you will need to update the value in TA0CCR0.

7. You want the LEDs to go off when there is no input, but there will always be a small amount of noise that lights up the LEDs. You can use an if/else statement to set TA0CCR1 to zero if the sampled value (after conditioning) is below a certain threshold, and set TA0CCR1 to the conditioned signal otherwise.

8. We can now connect our circuitry from labs 1 and 2 to the Launchpad. Use the Female-Male jumper wires to connect 3.3V and GND from the power supply to the Launchpad, connect the ADC input from the analog stage, and connect the PWM output to one of the LED banks. For now, use hookup wire to connect all of the inputs of the LED banks together. This will make all four LED banks flash in unison.

9. Debug and run your code. You should have a basic single channel color organ that responds to the amplitude of either the microphone or the line input at this point. If not, check steps VI-C-3 though VI-C-6 again to make sure that the signal is correctly conditioned. You can adjust either the amount of DC offset that is subtracted, or the noise threshold level in your if/else statement. Set these two variables such that the LED bank is off with no input, and lights up in response to an input. If the LED bank fails to light up, check the 4 Launchpad connections made in step 7 against the pinout of the MSP430G2553.

10. In addition to using the ***amplitude*** of the sampled signal to control the PWM output, we can also use the ***frequency*** of the sampled signal. This is accomplished with a block of code called a Lattice Wave Digital Filter. The code for a 3rd order filter is given below:

```
inp1 = voltageRaw;
inp2 = delay0;
p1 = inp1 - inp2;
outp2 = (alpha0 * p1) + inp2;
delay0 = outp2;
outp1 = outp2 - p1;
topout = outp1;

inp1 = delay1;
inp2 = delay2;
p1 = inp1 - inp2;
outp2 = (alpha2 * p1) + inp2;
delay2 = outp2;
outp1 = outp2 - p1;

inp1 = voltageRaw;
```



```
inp2 = outp1;
p1 = inp2 - inp1;
outp2 = (alpha1 * p1) - inp2;
delay1 = outp2;
outp1 = outp2 - p1;
botout = outp1;
```

11. A block diagram of this filter code is given on page 8 of the lab lecture notes. The input will be the **voltageRaw** variable sampled from the ADC10, and the output will be different for either a low pass output or a high pass output. Copy and paste the filter code given in step 10 after the first line in your while loop, and add the following line after the filter code:

```
lpfout = ((topout + botout)/2);
```

12. Update the amplitude conditioning code from steps 4-6 to use this new variable, **lpfout**, instead of the **voltageRaw** variable. It would be helpful to create a new temporary variable to do this.

13. Initialize the three delay variables in the filter code; **delay0**, **delay1**, and **delay2** to zero outside of the while loop.

14. The three alpha variables; **alpha0**, **alpha1**, and **alpha3** determine the crossover frequency of the filter. These variables can be calculated using a program in the Texas Instruments SLAA331 Application Report, but for now, use the following three alpha variables for a crossover frequency of 100 Hz, and add after the delay variable initialization:

```
alpha0 = 0.121715545654297;
alpha1 = 0.120417892932892;
alpha2 = 0.022688925266266;
```

15. Declare all of the new variables that you have added to you code with the type **float**. There are three delay variables, three alpha variables, and eight variables from the filter code.

16. Debug and run your code. If you missed any variable declarations, the compiler should let you know. You will have to add these before you proceed.

17. Test the functionality of the color organ with either the microphone or the line input. It should only respond to low frequency signals now, and ignore high frequency signals. You might have to readjust the amount of DC offset subtracted (step VI-C-4), or the noise threshold (step VI-C-7) to get the color organ to work correctly.

18. The filter code given in step VI-C-10 creates both a low pass filter and

a high pass filter. To implement the high pass filter, add the following line of code after the low pass line:

```
hpfout = ((topout - botout)/2);
```

19. We currently have Timer A0 setup to output the low pass filter signal. We can also use Timer A1 to output the high pass filter signal. Initialize Timer A1 to output TA1.1 at pin P2.1. Repeat steps VI-B-3 through VI-B-8 to initialize Timer A1 for the high pass filter.

20. Create a second amplitude conditioning code for the high pass filter, just as you did in steps VI-C-4 through VI-C-7 for the low pass filter. You will have to create another temporary variable, so you can adjust the DC offset and noise threshold of the high pass filter independent of the low pass filter.

21. Connect the Timer A1 high pass filter output TA1.1 to two of the LED bank outputs, and connect the Timer A0 low pass filter output TA0.1 to the remaining LED bank outputs.

22. Adjust the DC offset and noise threshold of the low pass and high pass filters until you get satisfactory performance using either the microphone input or the line input.

D) Final Demonstration

1. Verify that all components of the Digital Color Organ are functioning properly before starting the final demonstration.

2. Show that the LED outputs respond to the microphone input. The LEDs should turn off completely when there is no input.

3. Show that the LED outputs respond to the line input summer. Adjust the volume of the source, and pause/play the music.

4. Using either the microphone input or the line input summer, show that the low pass filter isolates low frequency sounds, and the high pass filter isolates high frequency sounds.

E) Clean up

Do not remove the scope probes from the scope. Turn off all equipment. Put all of your things back into your Storage Box and take it home with you.

VII. ASSIGNMENT FOR NEXT LAB PERIOD

1. Finish your Digital Color Organ Final Report.

Lab Report

Lab IV - Digital Color Organ - Analog-to-Digital Converters,
and Lattice Wave Digital Filters

Name: .....

Date: .....

Code of Ethics Declaration

All of the attached work was performed by me. I did not obtain any information or data from any other student. I will not post any of my work on the World Wide Web.

Signature .....

VI-D-2

Microphone Amplifier: _____

VI-D-3

Line Input Summer: _____

VI-D-4

Low Pass Filter: _____

High Pass Filter: _____