Design Assignment 3: Signal Conversion



Alexis Adie and Madison Mastroberte

ELC 411-01: Embedded Systems

Submitted:

October 26th 2017

Design Assignment 3: Signal Conversion

Alexis Adie and Madison Mastroberte

Department of Electrical and Computer Engineering

The College of New Jersey

2000 Pennington Road, Ewing, NJ 08618, USA

(adiea1, mastrom7)@tcnj.edu

# Introduction

The purpose of this assignment was to familiarize students with the Digital to Analog Converter (DAC) and the Analog to Digital Converter (ADC) available within PSoC Creator. Before lab, the team was required to complete the given ‘C’ code. The first bit of code, toggled the value of the DAC from ¼ and ¾ of its total value. The second bit required the input of an equation to calculate the voltage to be output to the LCD screen. In lab, the team used the ADC and DAC and recorded the subsequent values. Students then observed the output waveform and calculated the rise time and fall time. Overall the lab demonstrated further functionalities of the VDAC and ADC, and simulating it on the oscilloscope.

# II. Methodology

1. *Pre-Class*

Students generated code to drive the VDAC between two states, which were indicated by 0.25 and 0.75 of the value 255. Once calculated, it came to the values of 64 and 192. The next portion consisted of creating a line that converts the resulting SAR\_ADC result to a specific voltage.

1. *Part I: Analog to Digital Conversion*

Using the NI VirtualBench, student applied a voltage to the PSoC board and utilized the LCD conversion to find the ADC code equivalency to an input voltage. The codes were then recorded and plotted to demonstrate relationship and noise. Equation 1 was used to predict the code values outputted.

(1)

1. *Part II: Digital to Analog Conversion*

Students implemented the code in the Pre-Class portion, and measured the period of the signal and steady state voltages for the two drive states. Using Equation 1, students calculated the voltage corresponding to the quantization interval in regards to changes in the DAC code.

(2)

Using Equation 1, students calculated the voltage corresponding to the quantization interval in regards to changes in the DAC code. Error calculations were then evaluated using Equation 2.

(3)

Equation 2 calculates the error between the calculated and theoretical values for the code conversion. This process was completed in MATLAB, and the code is shown in Appendix A.

# III. Results

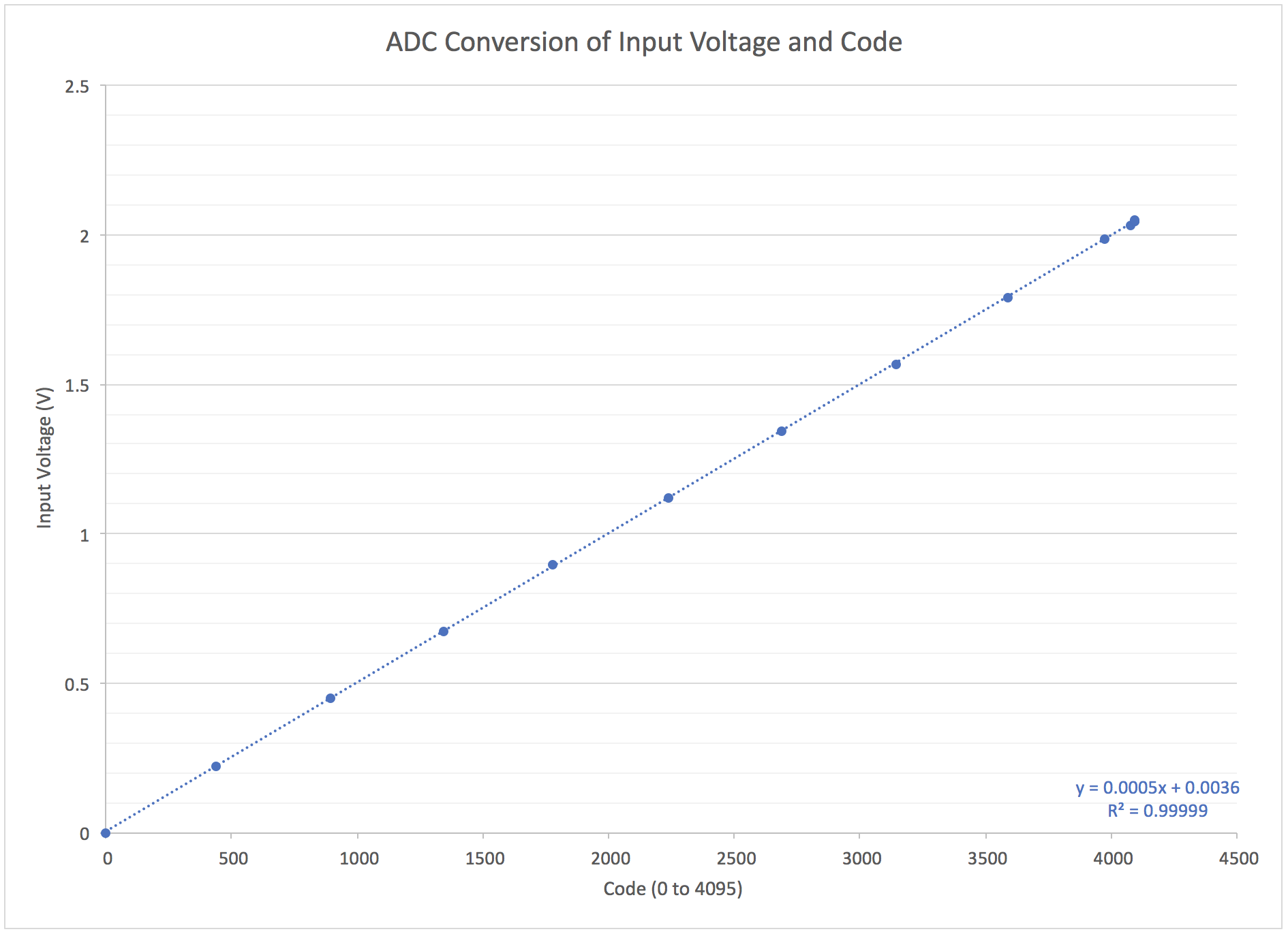
1. *Part I: Analog to Digital Conversion*

The ADC code values, interpreted voltage and input voltage are in the table below. The data is then plotted in Figure 1a, in terms of input voltage and ADC code. The relationship is linear, with a coefficient of determination of 0.99. However, this is an aggregate of all the data points and buffers any noise within the system. Therefore, noise must be shown in an additional graph.

**Table 1.** Values taken from LCD on PSoC 5LP from ADC conversion.

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Predicted Code** | **Volts** | **Multimeter** |
| 000 | 000 | 0.000 | 0.000 |
| 438 | 446 | 0.893 | 0.224 |
| 893 | 890 | 1.782 | 0.448 |
| 1342 | 1340 | 2.683 | 0.672 |
| 1777 | 1789 | 3.583 | 0.896 |
| 2239 | 2095 | 4.49 | 1.120 |
| 2689 | 2687 | 5.381 | 1.344 |
| 3143 | 3141 | 6.29 | 1.568 |
| 3589 | 3591 | 7.191 | 1.792 |
| 3975 | 3969 | 7.946 | 1.984 |
| 4074 | 4066 | 8.14 | 2.032 |
| 4092 | 4094 | 8.198 | 2.044 |
| 4095 | 4094 | 8.198 | 2.047 |
| 4095 | 4094 | 8.198 | 2.048 |

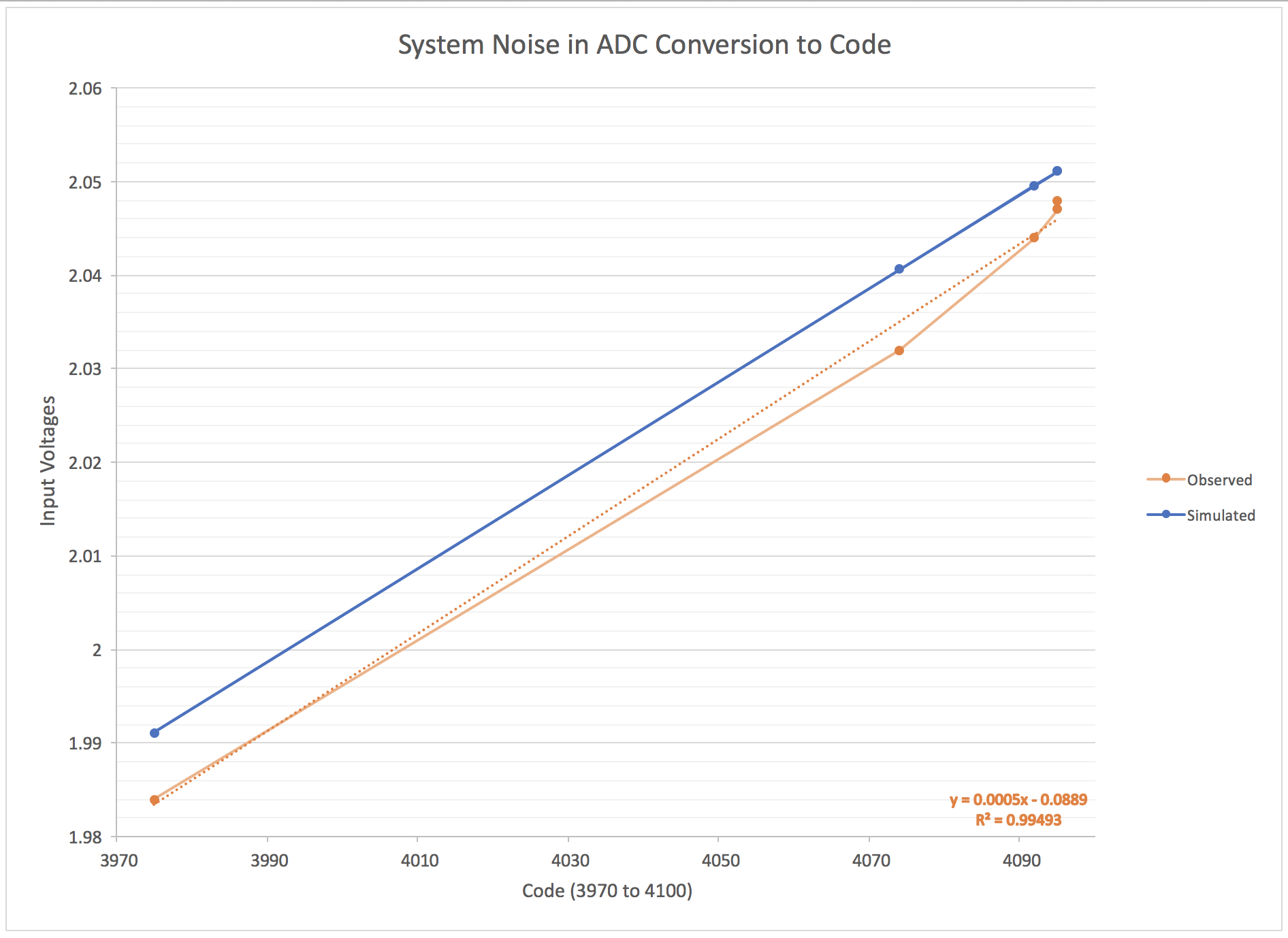
In Figure 1a, the graph shows the relationship between input voltage and code. The equation and coefficient of determination is shown on the bottom right.



**Figure 1a.** Graph of ADC conversion from input voltage to ADC code.

Although it difficult to distinguish, there is a cluster of points towards to end of the range, 4095. Since there is a cluster of data points, these were then plotted in Figure 1b. This graph features a smaller range and plots the actual ADC code values measured in the orange color. The equation for this is shown as well, however, the coefficient is lower since it is not as linear. The line in blue is the equation from the previous graph and averaged points to follow the linear trend.

In the figure below, the system noise is shown by demonstrating that there is a latency in changes at the beginning and end of the values range. In other words, numerous zero points and numerous 4095 points, since the system does not adapt quickly.



**Figure 1b.** Close-up of ADC conversion to show noise in system.

**Table 2.** Percentage and root-mean squared difference of ADC code.

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Predicted Code** | **Error**  **(%)** |  |
| 000 | 000 | 0 | 0 |
| 438 | 446 | 17.9 | 64 |
| 893 | 890 | .3 | 9 |
| 1342 | 1340 | .15 | 4 |
| 1777 | 1789 | .67 | 144 |
| 2239 | 2095 | 6.8 | 20736 |
| 2689 | 2687 | .07 | 4 |
| 3143 | 3141 | .06 | 4 |
| 3589 | 3591 | .05 | 4 |
| 3975 | 3969 | .05 | 36 |
| 4074 | 4066 | 1.9 | 64 |
| 4092 | 4094 | .04 | 4 |
| 4095 | 4094 | .02 | 1 |
| 4095 | 4094 | .02 | 1 |

The average percent error was 2.2%

Using equation 2, the root-mean-squared difference was calculated to be 145.2 rms.

1. *Digital to Analog Conversion*

In Figure 2a-b, the scope traces of the system are shown. Figure 2a shows the rising edge of the waveform, with two markers at the settling time to 0.1%, step 25% to 75%. Figure 2b shows the falling edge of the waveform, with two markers at the settling time to 0.1%, step 75% to 25%.



**Figure 2a.** Rising edge scope trace of waveform.



**Figure 2b.** Falling edge scope trace of waveform.

The values for the rise time and fall time are shown in Table 2. However, there a variety of difference in the different method types. Automatic values were found utilizing the NI measuring toolbox, the manual values were calculated using Equation 1, and the data sheet values were found utilizing the official PSoC 5LP manual.

Using equation 1:

*=*

**Table 3.** Various values for VDAC settling time on PSoC 5LP

|  |  |  |
| --- | --- | --- |
| Type | Rising Edge | Falling Edge |
| Automatic | 7.9847 s | 8.4531 s |
| Data Sheet | 3.2 s | 3 s |

Error=

*Error of rising edge =149%*

*Error of falling edge=181%*

The errors of the settling times were both over 100%. These errors may have come from defects within the probes used to carry the voltages, or from the computer itself lagging.

# IV. Discussion

During the experiment, the students noticed that while inputting a voltage, the code on the LCD display would jump between numbers. As a result, the team chose to record the mean values presented. This “noise” that was observed tended to only show codes a few numbers away from the most consistent numbers, often only jumping by one. By putting dividing each voltage by the each code, the team found that each code coincided with a voltage of about 0.002 V.

V. Conclusion

Through this project, students gained more experience implementing the DAC and ADC within PSoC Creator. The team was able to both measure and predict the ADC codes produced. By doing so, the relationship between voltage and the codes were found to be linear. Next, the rising and falling edges of the VDAC scope readings showed its settling time. While there were significant errors found during calculations, much can be accounted for due to computer lag issues.

# VI. Appendix

**Madison Mastroberte’s Code**

#include "project.h"

#include "stdio.h"

CY\_ISR\_PROTO(my\_isr);

CY\_ISR(my\_isr)

{

static int count = 0;

// must be static, so value retained between interrupts, one time initialization

// drives VDAC8 to 1/4 full scale and 3/4 full scale

if (count == 0)

{

VDAC8\_SetValue(64);

}

else

{

VDAC8\_SetValue(192);

}

CyDelay(500)

// Don't change code below

isr\_1\_ClearPending();

count = 1 - count; // toggle between 0 and 1

}

int

main( void )

{

int sar\_result;

// holds result from ADC\_SAR

int volts\_int;

// integer part of volts, for string formatting to overcome bug in PSoC Creator

int volts\_frac;

// fractional part of volts, for string formatting to overcome bug in PSoC Creator

char disp\_str[17];

// char array large enough to hold one line for display

double volts;

// student computes this, as a function of sar\_result

CyGlobalIntEnable; /\* Enable global interrupts. \*/

// Initialization, start your engines ...

ADC\_SAR\_Start();

ADC\_SAR\_StartConvert();

// Needed because ADC\_SAR is free-running

VDAC8\_Start();

isr\_1\_StartEx(my\_isr);

LCD\_Char\_Start();

// Loop forever

while (1)

{

sar\_result = ADC\_SAR\_GetResult16();

// get new ADC value

volts = (volts\_int \* sar\_result)/1023;

// Don't change anything below!

volts\_int = (int) volts;

// get integer part

volts\_frac = (int) ((volts - volts\_int) \* 1000 + 0.5);

// get fractional part as a 3-digit integer

// Display the string on top line, left justified

LCD\_Char\_ClearDisplay();

LCD\_Char\_Position(0, 0);

sprintf( disp\_str, "code=%4d", sar\_result);

LCD\_Char\_PrintString(disp\_str);

// Display the string on bottom line, left justified

LCD\_Char\_Position(1, 0);

sprintf( disp\_str, "volts=%d.%03d", volts\_int, volts\_frac );

LCD\_Char\_PrintString(disp\_str);

CyDelay(500);

// a little time for display to stabilize

}

}

**Alexis Adie’s Code**

#include "project.h"

#include "stdio.h"

CY\_ISR\_PROTO(my\_isr);

CY\_ISR(my\_isr)

{

static int count = 0;

// must be static, so value retained between interrupts, one time initialization

// drives VDAC8 to 1/4 full scale and 3/4 full scale

if (count == 0)

{

VDAC8\_SetValue(1/4);

}

else

{

VDAC8\_SetValue(3/4);

}

CyDelay(500)

// Don't change code below

isr\_1\_ClearPending();

count = 1 - count; // toggle between 0 and 1

}

int

main( void )

{

int sar\_result;

// holds result from ADC\_SAR

int volts\_int;

// integer part of volts, for string formatting to overcome bug in PSoC Creator

int volts\_frac;

// fractional part of volts, for string formatting to overcome bug in PSoC Creator

char disp\_str[17];

// char array large enough to hold one line for display

double volts;

// student computes this, as a function of sar\_result

CyGlobalIntEnable; /\* Enable global interrupts. \*/

// Initialization, start your engines ...

ADC\_SAR\_Start();

ADC\_SAR\_StartConvert();

// Needed because ADC\_SAR is free-running

VDAC8\_Start();

isr\_1\_StartEx(my\_isr);

LCD\_Char\_Start();

// Loop forever

while (1)

{

sar\_result = ADC\_SAR\_GetResult16();

// get new ADC value

volts = sar\_result\*(6/256);

// Don't change anything below!

volts\_int = (int) volts;

// get integer part

volts\_frac = (int) ((volts - volts\_int) \* 1000 + 0.5);

// get fractional part as a 3-digit integer

// Display the string on top line, left justified

LCD\_Char\_ClearDisplay();

LCD\_Char\_Position(0, 0);

sprintf( disp\_str, "code=%4d", sar\_result);

LCD\_Char\_PrintString(disp\_str);

// Display the string on bottom line, left justified

LCD\_Char\_Position(1, 0);

sprintf( disp\_str, "volts=%d.%03d", volts\_int, volts\_frac );

LCD\_Char\_PrintString(disp\_str);

CyDelay(500);

// a little time for display to stabilize

}

}

**Commented and Debugged Code**

#include "project.h"

#include "stdio.h"

CY\_ISR\_PROTO(my\_isr);

CY\_ISR(my\_isr)

{

// one time initialization of count used to toggle between values

static int count = 0;

// drives VDAC8 to 1/4 full scale and 3/4 full scale

if (count == 0)

{

VDAC8\_SetValue(64);

}

else

{

VDAC8\_SetValue(192);

}

isr\_1\_ClearPending();

// toggles between 0 and 1

count = 1 - count;

}

int

main( void )

{

// sar\_result holds result from ADC\_SAR

int sar\_result;

// holds integer part and fractional part of volts respectively

int volts\_int;

int volts\_frac;

// char array that holds exactly one line for display

char disp\_str[17];

// floating point voltage result

double volts;

/\* Enable global interrupts. \*/

CyGlobalIntEnable;

// Initializes the converters, ADC starts converting, and calling the function

ADC\_SAR\_Start();

ADC\_SAR\_StartConvert();

VDAC8\_Start();

isr\_1\_StartEx(my\_isr);

LCD\_Char\_Start();

// Loops forever

while (1)

{

// get new ADC value

sar\_result = ADC\_SAR\_GetResult16();

//converts sar\_result to the corresponding floating point voltage

volts = (2.048 \* sar\_result)/1023;

// get integer part of volts

volts\_int = (int) volts;

// get fractional part of volts as 3 digit int

volts\_frac = (int) ((volts - volts\_int) \* 1000 + 0.5);

// Display the string on top line, left justified

LCD\_Char\_ClearDisplay();

LCD\_Char\_Position(0, 0);

sprintf( disp\_str, "code=%4d", sar\_result);

LCD\_Char\_PrintString(disp\_str);

// Display the string on bottom line, left justified

LCD\_Char\_Position(1, 0);

sprintf( disp\_str, "volts=%d.%03d", volts\_int, volts\_frac );

LCD\_Char\_PrintString(disp\_str);

//delay to allow display to stabilize

CyDelay(500);

}

}