IT327 Lab 9

Analog to Digital Conversion (ADC) and Digital to Analog Conversion (DAC)

Objective:

In this lab there are two main objectives, 1) to gain experience with and apply the theory of Digital to Analog Converters (DAC) and Analog to Digital Converters (ADC) and 2) to analyze the recreation of the analog signal by the DAC in the time domains.

You will apply theory by making predictions as to the outcome of the signal after it has passed through each converter. At the close of the lab, you should have in your write-up the calculations and tables, including a spreadsheet, and a minimum of 3 oscilloscope images.

Setup and Preparation:

This lab runs on the Cerebot MX7cK board, which uses a Microchip PIC32 microprocessor. There is a program written for each section of the lab. The three sections are analog to digital, digital to analog, and a combination of the two.

The board must have the proper program loaded for each section to work. The program is loaded on the board with the MPIDE program located in:

C:\Cerebot

Right click on mpide.exe and select "Run as administrator." Then click on the File menu, Sketchbook, and there should be ADC, DAC and ADC-DAC in the list. If they are not there click on the File menu, Preferences, and set the Sketchbook location to:

C:\Cerebot\sketchbook

MPIDE must also be configured for the correct board and COM port. This information is displayed on the lower right corner of the MPIDE program. If the board is incorrect click the Tools menu, Board, and select Cerebot MX7cK. The correct COM port can be discovered through Device Manager (listed as USB Serial Port) and set through the Tools menu, Serial Port, and selecting the correct COM port.

In addition to the board there is a potentiometer (pot) for the A-D input and a set of switches for the D-A input. There are also two P_{mods} (small auxiliary boards that attach to the MX7cK board); one with eight LEDs (8LD) and one with a DAC chip (DA1). The potentiometer should be plugged into JA, the 8LD P_{mod} into JB, the switches into JC, and the DA1 P_{mod} into the upper pins of JD.

Communication and power for the board is provided through the microUSB port at the lower left corner of the board closest to the switch.

Connect a voltage meter to the test points next to the potentiometer.

In Excel, create a table to predict the outcome of DAC or ADC by multiplying the step size by the decimal value of the digital input:

$\frac{\quad}{\quad}$ where V_{max} =3.3V and n = 8 bits

Decimal value of bits	Binary value	Analog Voltage	
0	00000000	0.000V	
1	00000001	0.013V	
2	00000010	0.026V	
…	…	…	
254	11111110	3.274V	
255	11111111	3.287V	Hint: use DEC2BIN function in Excel

Section 1 – Analog to Digital Conversion

Procedure:

Be sure to take plenty of pictures and include them in your write-up with an appropriate caption.

1. On the MPIDE program click the File menu, Sketchbook, and select ADC. This will open another window with the code to capture the analog input and change it into an 8-bit digital value which is displayed on the LED P_{mod} with the most significant bit (MSB) on the leftmost LED. The binary digits 0 and 1 are represented with the LED off and on respectively.

   The board provides 3.3 volts which are fed through a potentiometer that acts as a voltage divider that varies the input from 0 to 3.3 volts. The voltage increases with clockwise rotation of the potentiometer and can be read on the volt meter connected to the test points.

2. To load the program onto the board click the Upload button:

   

   This will compile the program and upload to the board. There is a progress bar that will stall near the end waiting to upload the program to the board. At this time either turn the board on or press the reset button on the board (next to JD at the top right corner) and the board will reboot and start listening on the COM port. Flashing red LEDs indicate that the board is receiving the program. Wait for this process to finish. Once uploaded the program is persistent on the board and will last through power loss.

3. Note the digital predictions for the analog values 0.309, 1.250, 2.604, and 3.004 volts. To predict these values, choose the digital output that is closest to your value. Now test your predictions by setting those voltages with the potentiometer. Record the binary output and its corresponding value in volts and compare to your predictions.

Analog V	Predicted (Dec #)	Binary Value	Actual (Dec #)
0.309			
1.250			
2.604			
3.004			

Conclusion: In addition to the follow questions, add your own conclusion as instructed in the syllabus:

Why can you not get the exact value?

How would you be able to better represent your exact value digitally?

Section 2 – Digital to Analog Conversion

Procedure:

1. On the MPIDE program click the File menu, Sketchbook, and select DAC. This will open another window with the code to capture the digital input from the eight switches and change it into an analog voltage output that varies between 0 to 3.3 volts which can be read from pins A1 (or B1) and ground of the DA1 P_{mod}. The leftmost switch represents the MSB. The switch is turned on and represents a binary 1 when up.
2. Follow the previous instructions provided in Section 1 to upload the DAC program to the board.
3. Choose 4 different digital inputs with the 8 bit switch and predict the analog output before you do it. Having produced the spreadsheet, this should also be easy. Now test your predictions with the DAC.

Predicted			Actual	
Binary Value	**Dec #**	**V out**	**Dec #**	**V out**

Conclusion: In addition to the follow questions, add your own conclusion as instructed in the syllabus:

Was it easier to predict the outcome this time?

What does this say about the endurance of digital information compared to analog?

Section 3 – ADC-DAC

Procedure:

1. On the MPIDE program click the File menu, Sketchbook, and select ADC-DAC. This will open another window with the code combined from the two previous sections that is designed to take an analog waveform as input, sample points on the waveform and convert the sample into digital information, then take the digital information and convert back to analog waveform.

   IMPORTANT: the board can be damaged if more than 3.5 volts is provided

   The waveform should be a sine wave provided by a function generator using DC offset so the waveform is between 0 and 3 volts. It is important to make sure the waveform does NOT go below ground (0V).

2. Follow the previous instructions provided in Section 1 to upload the ADC-DAC program to the board.

3. For the analog input, connect the function generator to pin 1 and ground of JA. To transfer the digital information from the ADC to the DAC jumper from JB to JC. Then connect the DA1 P_{mod} to the upper pins of JD. The output of DA1 between pins A1 (or B1) and ground should be connected to an oscilloscope.

   The program is a composition of the previous sections and loops through the code to capture the analog value, convert it to an eight individual bits which are passed through the jumpers, and then the eight bits are combined and passed to the DA1 P_{mod}. The time required to complete this loop is what sets the sample rate for the system. Each of the loops takes 40 µs which means the sampling rate is 25,000 samples per second.

   Look at the reconstructed waveform from the DAC to see how it compares to the original. Do this for frequencies of 100Hz, 500Hz, and 2kHz. Compare the oscilloscope images and analyze your results.

Conclusion: In addition to the follow questions, add your own conclusion as instructed in the syllabus:

Add to your own conclusions the following:

What determines how accurate an ADC will represent the original signal?

Why was the signal distorted at the higher frequency on Procedure 4?

Code:

ADC:

```
/*                    Analog to Digital Converter
                      Aaron Vivian - 3/29/2013
The analog input is provided by the reference voltage of 3.3 volts from the
system board running through a 1 megohm potentiometer acting as a
voltage divider.  This varies the voltage provided to the analog input pin 1
on JA.

The digital output is represented by a set of eight LEDs (Pmod 8LD)
connected to pins 8-15 on JB.  The most significant bit (MSB) is the
leftmost LED.
*/

// Constants setting the pin numbers for the LEDs and analog input.
const short LEDPin[] = {8, 9, 10, 11, 12, 13, 14, 15};
const short analogPin = 0;
// Variables storing the analog input value and the digital bits.
short analogValue = 0;
short binaryNumber[8];

void setup() {
  // Initialize the eight pins on JB as outputs.
  for (short i = 0; i < 8; i++) {
  pinMode(LEDPin[i], OUTPUT);
  }
  // Initialize the analog input pin.
  pinMode(analogPin, INPUT);
}

void loop() {
  // Read the value and divide by 4 to get from ten to eight bits.
  analogValue = analogRead(analogPin)/4;
  /* Iterate through the binaryNumber array storing the least significant bits
  to most significant bits.  The bits are obtained by taking the modulus of
  analogValue and 2 then doing integer division of analog value and 2
  which truncates the answer to the integer.
  */
  for (short i = 7; i >= 0 ; i--) {
    binaryNumber[i]= analogValue % 2;
    analogValue = analogValue / 2;
  }
  // Iterate through the binaryNumber array and turn the LEDs on or off
  // depending on the value.
  for (short i = 0; i < 8; i++) {
              if (binaryNumber[i] == 1) {
               digitalWrite(LEDPin[i], HIGH);
              } else {
               digitalWrite(LEDPin[i], LOW);
              }
  }
}
```

DAC:

```
/*                    Digital to Analog Converter
                      Aaron Vivian - 3/29/2013
Input switches in series with 4.7k ohm resistors provide the eight digital bits
connecting to JC.  Based on Digilent Switch Module (Pmod SWT).

Output to DAC module (Pmod DA1) plugged into upper pins of JD.  The
output on DA1 is on pin A1 or B1 of the module at 0 to 3.3V.
 */

#include <DACSPI1.h>
#include <DSPI.h>

// Constants setting the pin numbers for the switches.  The switches are
// mapped to the correct pins here.
const short switchPin[] = {21, 20, 23, 22, 18, 19, 17, 16};
// The array to hold and process the switch states.
short compositeArray[8];
// The variable to hold the combined value of the eight switch states.
int compositeValue = 0;
// Instantiate the library object.
DACSPI1 newDACSPI1;

void setup() {
  // Serial output for debugging.
  Serial.begin(9600);
  // Initialize PmodDACSPI1.
  newDACSPI1.begin(PAR_ACCESS_SPI0);
  // Initialize the eight pins on JC (17-24) as inputs.
  for (short i = 0; i < 8; i++) {
    pinMode(switchPin[i], INPUT);
  }
}

void loop(){
  // This for loop reads the state of each switch.  It then enters a value of 0 or
  // 1 into the appropriate location in compositeArray.
  for (short i = 0; i < 8; i++) {
   if (digitalRead(switchPin[i]) == 0) {
     compositeArray[i] = 0;
   } else {
    compositeArray[i] = 1;
   }
  }
  // This loop performs an OR function with the value in the array and
  // compositeValue and then shifts the bits to the left by one to create
  // compositeValue which is then passed to the DA1.
  for (short i = 0; i < 8;  i++) {
    compositeValue = compositeArray[i] | compositeValue;
    if (i < 7)
       compositeValue = compositeValue << 1;
  }
  // This writes compositeValue to DA1.
  newDACSPI1.WriteIntegerValue(compositeValue);
  // Debugging output.  Loops through the array and prints the 1 or 0 for
  // each bit then prints compositeValue.
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  for (short i = 0; i < 8; i++) {
    Serial.println(compositeArray[i]);
  }
  Serial.println();
  Serial.println(compositeValue);
  Serial.println();
  delay (1000);
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  // Resetting compositeValue to 0 each time through the loop.
  compositeValue = 0;
}
```

ADC-DAC

```
/*                    A-D D-A Converter
                      Aaron Vivian 3/29/2013
This is the code from the ADC and DAC combined to show a stepped
waveform.

The input is on pin 0 (marked 1 on JA) and ground.  The output is either A1
or B1 on the DA1 module plugged into the upper pins of JD.

The pins between JB and JC are jumpered.

The code is copied from the ADC and DAC less the comments and
modified for correct pin mapping.
*/

#include <DACSPI1.h>
#include <DSPI.h>

const short analogPin = 0;
const short LEDPin[] = {8, 9, 10, 11, 12, 13, 14, 15};
const short switchPin[] = {20, 21, 22, 23, 16, 17, 18, 19};
short analogValue = 0;
short binaryNumber[8];
short compositeArray[8];
int compositeValue = 0;
DACSPI1 newDACSPI1;

void setup() {
  pinMode(analogPin, INPUT);
  for (short i = 0; i < 8; i++) {
   pinMode(LEDPin[i], OUTPUT);
  }
  for (short i = 0; i < 8; i++) {
    pinMode(switchPin[i], INPUT);
  }
  newDACSPI1.begin(PAR_ACCESS_SPI0);
}

void loop() {
  analogValue = analogRead(analogPin)/4;
  for (short i = 7; i >= 0 ; i--) {
  binaryNumber[i]= analogValue % 2;
  analogValue = analogValue / 2;
  }
  for (short i = 0; i < 8; i++) {
                if (binaryNumber[i] == 1) {
                 digitalWrite(LEDPin[i], HIGH);
                } else {
                 digitalWrite(LEDPin[i], LOW);
                }
  }
  for (short i = 0; i < 8; i++) {
   if (digitalRead(switchPin[i]) == 0) {
     compositeArray[i] = 0;
   } else {
    compositeArray[i] = 1;
   }
  }
  for (short i = 0; i < 8;  i++) {
    compositeValue = compositeArray[i] | compositeValue;
    if (i < 7)
       compositeValue = compositeValue << 1;
  }
  newDACSPI1.WriteIntegerValue(compositeValue);
  compositeValue = 0;
}
```