# 

# Analog Conversion

***C8051 Microcontroller***

Jeffrey Pistacchio

Christopher Pybus

Microprocessor Systems

31 October 2016

# Introduction

Converting signals between digital and analog media is the basis for connecting the real world to electronic data. Thus, being able to make theses conversions is essential for every field of study that requires computers to make measurements or generate data. This lab demonstrated how conversion between digital and analog signals is conducted and the applications thereof in three parts. Part 1 included making a voltmeter. (Skip Part 2) Part 3 required outputting a generated digital signal as an analog signal. Part 4 combined the previous two parts by inputting an analog signal, converting it into digital data, applying a digital notch signal, and outputting the filtered signal.

# Methods and Procedures

Each part of the lab was divided into a number of individual goals. Part one was divided into three goals: wiring a potentiometer so that the variable voltage across it can be read by the C8051 microprocessor, writing software to be able to convert from analog to digital data, and formatting the program to display along with the current read voltage, the highest, lowest, and average voltage read. Part three was divided into two goals: generating and outputting sawtooth and sine wave signals using digital to analog conversion, and being able to output a sine wave that was inputted to the C8051 from a function generator. These signals needed to be displayed on an oscilloscope. Part four was one goal: use the multiply and accumulate (MAC) registers to execute a simple IIR filter on the inputted signal. There were other objectives in part four, but they have already been accomplished in previous parts and do not need further explanation.

## Part 1 - Goal 1

This section was relatively straightforward. By connecting a potentiometer between power and ground, the voltage across the potentiometer can be controlled by twisting the potentiometer. The output of the potentiometer was connected to the crossbar input on pin 47, which is the AIN0.0 port. See Appendix A for further wiring details.

## Part 1 - Goal 2

The REF0CN register, the reference voltage was set to the internal 2.4V. Using registers AMX0CF and AMX0SL, AIN0.0 (pin 47 on the EVB) was set to single ended analog input. The SAR clock and gain were set to 2 MHz and 1, respectively, using the ADC0CF register. In the main code every iteration of a while loop took an analog measurement using the read\_AD\_input() function. This function starts a conversion by setting AD0BUSY high and waiting for the flag AD0INT. Finally the function returns the 12 bit digital value.

## Part 1 - Goal 3

Recording the high and low measurements was accomplished using if statements and a saved value. If the ADC value read was lower or higher than the current records, then it would replace the current record with the new value. Average value was accomplished by averaging every voltage read using the MAC on the 8051 microcontroller. The MAC provides an easy and fast way to multiply and add large numbers, which is why it was used in this scenario. Because it is so fast, the team was able to take more samples per second (relative to using regular math functions).

## Part 3 - Goal 1

Outputting digital data using DAC0 required configuring the DAC0CN register. This was configured in a way that allowed DAC updates to occur whenever on a write to the register DAC0H. This is done in the function dac\_output(val), where val is the short int containing the voltage level needed to be outputted. The sawtooth wave was generated by writing an incrementing variable to the DAC. The team also generated a sine wave to the DAC using math and a for loop in the software.

## Part 3 - Goal 2

Outputting the data on the DAC that was originally inputted on the ADC pin was accomplished by making an A/D conversion and then immediately sending the converted data to the DAC to be outputted back out on the board. This is fairly simple considering the data entering was in the same format as the data exiting. However, the team had trouble when inputting a sine wave from the function generator, as half of the ADC readings were negative voltages, and the ADC was not equipped to handle negative AC voltage. This was solved by offsetting the input to all positive voltage.

## Part 4 - Goal 1

The team used the MAC to execute quick math operations on the voltages being read by the ADC in order to create an IIR filter. A total of four MAC operations were needed for each filter pass. Previous voltage readings and filter results needed to be stored for future filter passes. The MAC was set to operate in fractional mode so it would function with the fractional coefficients on each term. The equation used for the filter is shown in figure 1 below, and the simplified version is shown in figure 2 below.



*Figure 1 - Original IIR filter equation*



*Figure 2 - Simplified version of IIR filter equation*

The MAC takes in two values for each operation, multiples them, and then adds the result to whatever is currently in the accumulator. The operation flow for this part is listed below.

1. Clear accumulator and reset the MAC
2. Multiply 10/32 to current voltage reading, add result to accumulator
3. Multiply 25/104 to previous voltage reading, add result to accumulator
4. Multiply 10/32 to second previous voltage reading, add result to accumulator
5. Multiply 19/64 to previous filter result, add result to accumulator
6. Read accumulator value
7. Update previous values with new values
8. Output filter result to DAC

## Part 4 - Miscellaneous

Part 3 also involved inputting and outputting analog values, but that was already accomplished in the first part as as such, it was not described again. A simple change that should be noted is the justification of the analog input and output was changed to be right justified instead of left justified. This made it so the bits were already in the correct positions when they entered and exited the filter, and no further bit shifts were needed thus saving time.

# Results and Analysis

All parts were tested by visually verifying the results in the provided oscilloscope. For example, in Part 1, a sawtooth function needed to be outputted by the DAC. The team connected the oscilloscope to the output and could visually see that the function was incrementing in a stair-stepping motion.

For part 3, the team connection the oscilloscope to both the output from the DAC and the input from the function generator. That way, they were able to see that the sine wave that the function generator was identical to the function that the DAC was outputting.

For part 4, the team used the same method. The function generator was connected to the oscilloscope so the team could see both the DAC output and the ADC input from the generator. However, for this part, the waves should begin by being identical, but as frequency increases the DAC output wave got smaller and smaller until it hit the notch around 19 kHz. After that it began to increase in amplitude.

# Conclusion

The team successfully accomplished all parts of Lab 4 by making several smaller programs that accomplished all the subgoals. Dividing the project into separate goals made the exercises significantly more approachable due to implementing the divide and conquer mindset. Each goal was simple enough that each group member could complete a goal by the time they came to class, and spend lab time debugging code and building hardware.

Given more time the team would have liked to add enhancements, such as using differential input on the ADC so they could handle negative input. In addition to the issue of time, the team also recognizes that going to find help earlier would have remedied many issues. Furthermore, paying attention to details in the code and better commenting practices would have led to catching mistakes earlier and being better able to finish the lab on time.

# Appendix A - Schematic for Part 1