8-bit Digital to Analog Converter Testing

Jason Beaulieu

Abstract

An 8-Bit Current Steering DAC operating from three supplies of 5v, 2.5v and 3.5v will be tested. The 8-bit binary input works with 0-5v logic, and the clock can be as high as 25MHz. The common-mode voltage Vcm runs from a 3.5v rail, and the positive input of the Op Amp from a Vref = 2.5v reference. The design was manufactured in a 0.5μm AMI C5N process.

Contents

I. TEST PROCEDURES

The following section discusses the procedures performed for testing of the physically implemented chip.

A. Device Pin Out

The following figure shows a pin out of the chip.

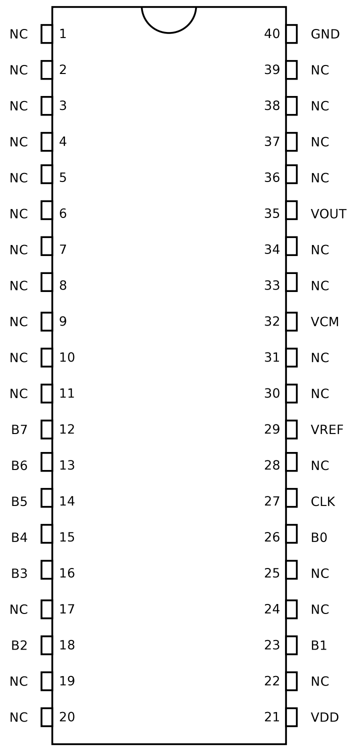

Fig. 1. Package Pinout.

B. Test 1: Static Power Dissipation

This test is a simple test to determine if there are power supply shorts to ground.

- **Procedure**
  - Tie output pins low with dummy load of 100kΩ, and input pins to ground.
  - Connect Vref and Vcm supplies to their respective pins.
  - Ramp VDD supply to 5V, observing current draw from power supply.
  - If current draw is excessive, it most likely means shorts in circuit design or faulty device. If similar results are found on multiple devices suspect a short in design.

- **Results**
  - The device draws 0mA of current from 5V supply. This results in a idle power draw of 0mW.
  - This is resonable since when the circuit is in steady state the only load should be the opamp, which only pulls 100μA and was smaller than the meter could read.

C. Test 2: DAC Operation

This test verifies the DAC design.

- **Procedure**
  - Connect Vref and Vcm supplies to their respective pins.
  - Apply 0-5V binary input signal to input pins.
  - Vary binary input signal frequency.
  - Observe output of DAC on oscilloscope. A low pass filter may be applied to output to smooth edges.
- **Results**
  - **Binary Ramp:** The following figure shows the schematic used to test the response of the DAC to binary ramp. The input voltage source is running a 0-5v square wave at 2MHz. Figure 3 shows a oscilloscope screen capture of the node V_{out}.

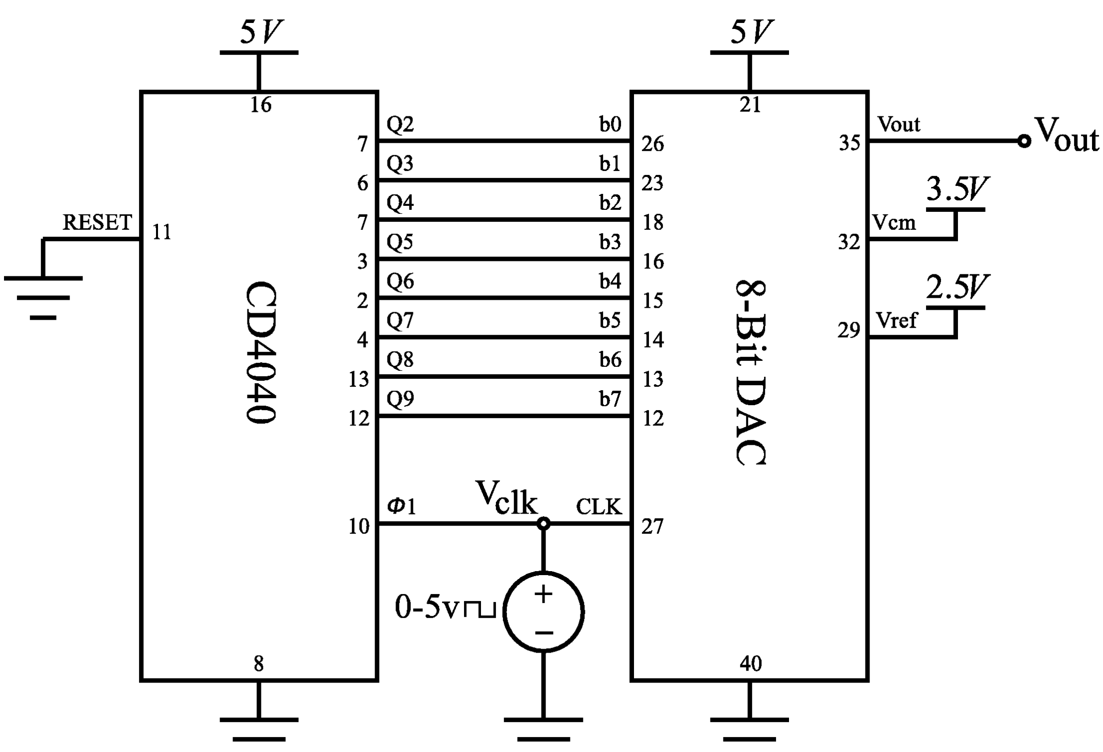

Fig. 2. DAC Ramp Output.

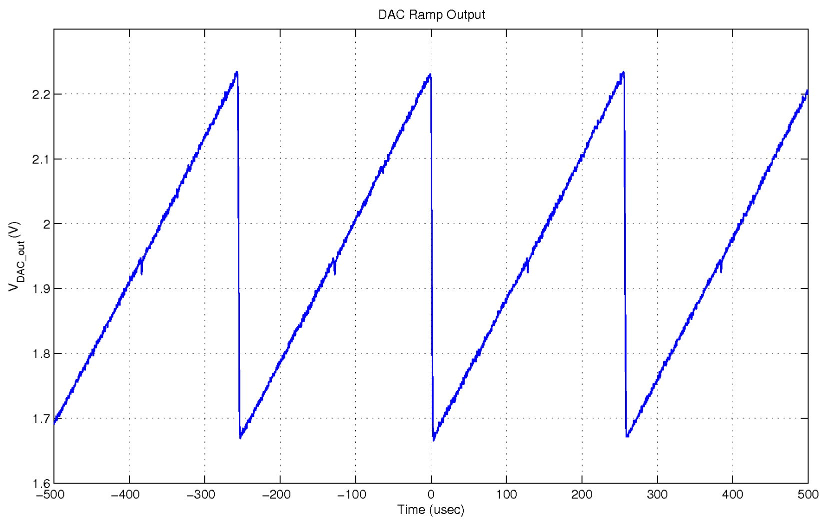

Fig. 3. DAC Ramp Output.

The output looks pretty linear through all possible output combinations. The small glitches that occur at the major bit transitions were seen during simulation and are a result of the latches used in the design. When the input frequency is increased to greater than 2MHz the glitches become larger and more can be seen. 2MHz seemed to be a decent operating frequency.

– **Binary Sinewave:** The following figure shows the schematic used to test the response of the DAC to a binary sinewave. The input voltage source is running a 0-5v square wave at 15MHz. The source code for the PIC16F628A is in Appendix A.

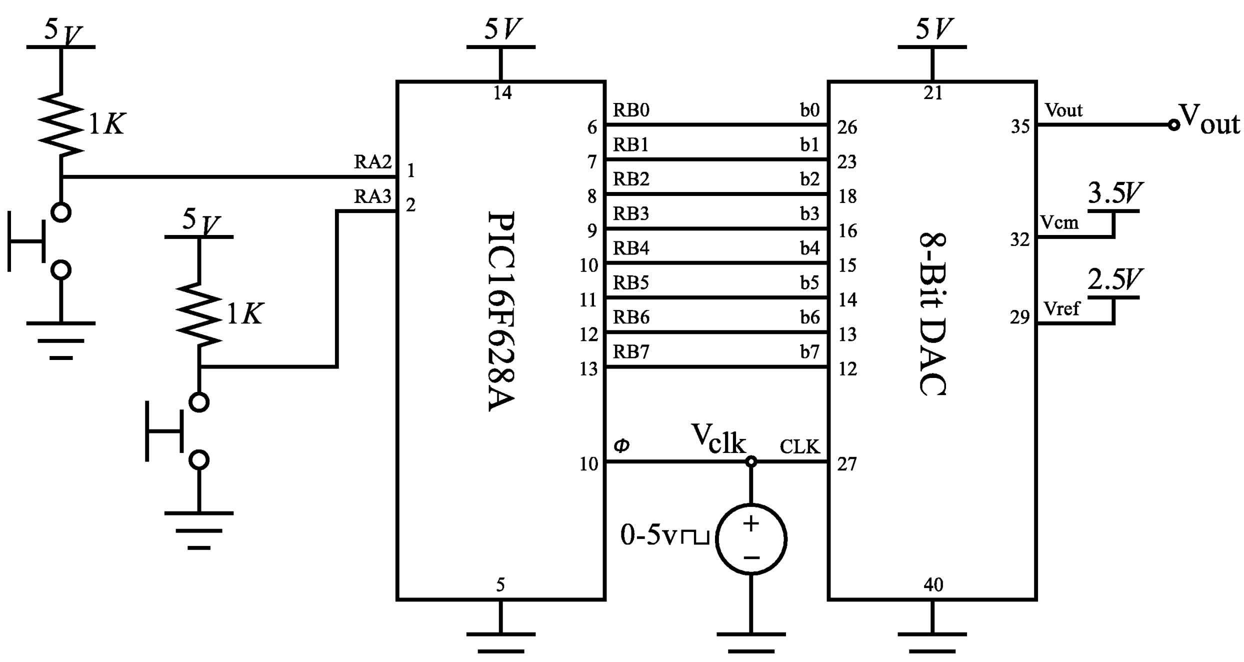

Fig. 4. DAC Ramp Output.

Figure 5 is an oscilloscope screen capture of the DAC output.

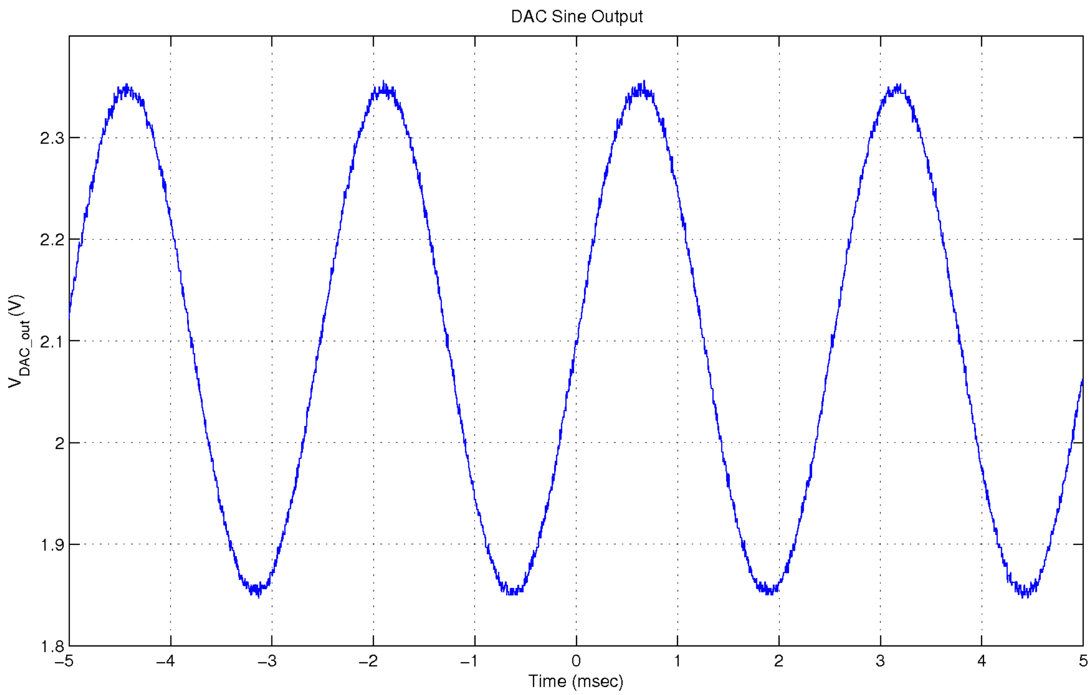

Fig. 5. DAC Sine Output.

The sinewave output looks very good. One thing to notice is the amplitude of the signal, it only goes from 1.85v to 2.35v, but the simulations showed the output range being 1.5v-3.5v. The reason for this was not explored, but could be attributed to an inaccurate simulation model, or a fault in the manufacturing process.

II. Appendix A

- Source code for 8-bit_sine.c.

```c
/*Jason Beaulieu*/
/*Description:  Runs through a sequence of binary values to create a sine wave using a
DAC*/
/* to compile: picl -16f627 8-bit_sine.c */
#include <pic.h>

#define _buttonOn       0
#define _button1        RA2
#define _button2        RA3
#define _debounce       20000

char i=0;
char inc=1;
long int j;

const int buf[256] = {0xf2, 0xf2, 0xf2, 0xf1, 0xf1, 0xf0, 0xf0, 0xf0, 0xef, 0xef,\
 0xee, 0xed, 0xed, 0xec, 0xeb, 0xea, 0xe9, 0xe8, 0xe7, 0xe5, 0xe4, 0xe3, 0xe1, 0xe0,\
 0xde, 0xdd, 0xdb, 0xd9, 0xd8, 0xd6, 0xd4, 0xd2, 0xd0, 0xce, 0xcc, 0xca, 0xc8, 0xc6,\
 0xc4, 0xc1, 0xbf, 0xbc, 0xba, 0xb8, 0xb5, 0xb3, 0xb0, 0xae, 0xab, 0xa9, 0xa6, 0xa3,\
 0xa0, 0x9e, 0x9b, 0x98, 0x96, 0x93, 0x90, 0x8e, 0x8b, 0x88, 0x85, 0x82, 0x7f, 0x7c,\
 0x7a, 0x77, 0x74, 0x72, 0x6f, 0x6c, 0x69, 0x66, 0x64, 0x61, 0x5e, 0x5c, 0x59, 0x56,\
 0x54, 0x51, 0x4f, 0x4c, 0x4a, 0x47, 0x45, 0x43, 0x40, 0x3e, 0x3b, 0x39, 0x37, 0x35,\
 0x33, 0x31, 0x2f, 0x2d, 0x2b, 0x29, 0x27, 0x25, 0x24, 0x22, 0x21, 0x1f, 0x1e, 0x1c,\
 0x1b, 0x1a, 0x18, 0x17, 0x16, 0x15, 0x14, 0x13, 0x12, 0x11, 0x11, 0x10, 0x10, 0x0f,\
 0x0f, 0x0e, 0x0e, 0x0e, 0x0d, 0x0e, 0x0d, 0x0e, 0x0e, 0x0d, 0x0e, 0x0e, 0x0f, 0x0f,\
 0x10, 0x10, 0x11, 0x12, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x1a, 0x1b, 0x1c,\
 0x1e, 0x1f, 0x21, 0x22, 0x24, 0x25, 0x27, 0x29, 0x2b, 0x2d, 0x2f, 0x31, 0x33, 0x35,\
 0x37, 0x39, 0x3c, 0x3e, 0x40, 0x42, 0x45, 0x47, 0x4a, 0x4c, 0x4f, 0x51, 0x54, 0x56,\
 0x59, 0x5c, 0x5f, 0x61, 0x64, 0x67, 0x69, 0x6c, 0x6f, 0x72, 0x74, 0x77, 0x7a, 0x7d,\
 0x7f, 0x82, 0x85, 0x88, 0x8b, 0x8d, 0x90, 0x93, 0x95, 0x99, 0x9b, 0x9e, 0xa1, 0xa3,\
 0xa6, 0xa9, 0xab, 0xae, 0xb0, 0xb3, 0xb5, 0xb8, 0xba, 0xbd, 0xbf, 0xc1, 0xc4, 0xc6,\
 0xc8, 0xca, 0xcc, 0xce, 0xd0, 0xd2, 0xd4, 0xd6, 0xd8, 0xda, 0xdb, 0xdd, 0xde, 0xe0,\
 0xe2, 0xe3, 0xe4, 0xe5, 0xe6, 0xe8, 0xe9, 0xea, 0xeb, 0xec, 0xed, 0xed, 0xee, 0xef,\
 0xef, 0xf0, 0xf0, 0xf1, 0xf1, 0xf1, 0xf1, 0xf2};

main()
{
	//Initializations
	OPTION=0x88;
	CMCON = 0x07;
	TRISA=0b11111111;
	PORTA=0;
	TRISB=0b00000000;
	PORTB=0;
	while(1){
		if(_button1==_buttonOn || _button2==_buttonOn) {
			if(_button1==_buttonOn) {
				inc++;
			}
			if(_button2==_buttonOn) {
				inc--;
			}
			j=0;
			while(j<_debounce){
				j++;
			}
		}
		PORTB = buf[i];
		i=i+inc;
	}
}
```